UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA TÉCNICA DE TELECOMUNICACIÓN

DESARROLLO DE UN DATALINK CON EL CHIP CC1110 DE TEXAS INTRUMENTS PARA UAVs

Autor: David Rubio Vela

Tutor: Francisco José Arqués Orobón

E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Estudio y desarrollo de aplicaciones para un datalink de baja frecuencia

TÍTULO: DESARROLLO DE UN DATALINK CON EL CHIP CC1110 DE TEXAS INTRUMENTS PARA UAVs

AUTOR: DAVID RUBIO VELA

TUTOR: FRANCISCO JOSÉ ARQUÉS OROBÓN **Vº Bº.**

DEPARTAMENTO: DIAC

Miembros del Tribunal Calificador:

PRESIDENTE: AURELIO BERGES GARCÍA

VOCAL: FRANCISCO JOSÉ ARQUÉS OROBÓN

VOCAL SECRETARIO: CARLOS CORTÉS ALCALÁ

DIRECTOR:

Fecha de lectura: 25/09/2012

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

Se trata de estudiar el comportamiento de un sistema basado en el chip CC1110 de Texas Instruments, para aplicaciones inalámbricas. Los dispositivos basados en este tipo de chips tienen actualmente gran profusión, dada la demanda cada vez mayor de aplicaciones de gestión y control inalámbrico. Por ello, en la primera parte del proyecto se presenta el estado del arte referente a este aspecto, haciendo mención a los sistemas operativos embebidos, FPGAs, etc.
En una segunda parte se realiza el estudio del dispositivo mediante una placa de desarrollo, verificando y comprobando mediante el software suministrado, el alcance del mismo. Cabe resaltar en este punto que el control con la placa mencionada se debe hacer mediante programación de bajo nivel (lenguaje C), lo que aporta gran versatilidad a las aplicaciones que se pueden desarrollar. Por ello, en una tercera parte se realiza un programa funcional, basado en necesidades aportadas por la empresa con la que se colabora en el proyecto (INDRA).
Para terminar, se plantean lineas futuras de aplicación del sistema.

Índice

Agradecimientos 1

Abstract 2

Resumen 3

1. Presentación del proyecto: 4

2. Introducción a los UAVs 6

2.1. Historia 6

2.2. Componentes de un UAVs 7

2.2.1. Segmento aéreo: 7

2.2.2. Segmento terrestre: 7

2.3. Clasificación de los UAVs 8

2.3.1. Según la misión: 8

2.3.2. Según las tareas a realizar: 9

2.4. ISR (Inteligencia, vigilancia y reconocimiento) 9

3. Características de los sistemas embebidos: 11

3.1. Definición: 11

3.2. Hardware: 11

3.3. Software: 14

3.4. Aplicaciones: 15

3.5. Arquitecturas de computadores: 16

3.5.1. Introducción: 16

3.5.2. Micro's: 16

3.5.3. Comunicaciones: 17

3.5.4. Visualización: 17

3.5.5. Actuadores: 18

3.5.6. Pines I/O analógicos y digitales: 18

3.5.7. Reloj: 19

3.5.8. Modulo de alimentación (Power): 19

3.5.9. Generalidades sobre dispositivos CPU: 20

4. Transcurso del proyecto: 21

4.1. Programación básica en C del datalink: 22

5. Separación datalink genérico con la funcionalidad específica: 30

5.1. Modo Localización(Location o GPS Mode) 31

5.1.1. Características principales receptor GPS:.....33
5.1.2. Programación Modo Localización (Location o GPS Mode):.....34
5.1.3. Prueba de Modo Localización (Location o GPS Mode): 47
5.2. Modo Conversación Chat (User Mode).....73
5.2.1. Programación Modo Conversación Chat (User Mode) 73
5.2.2. Prueba Modo Conversación Chat (User Mode) 77
5.3. Modo En Espera (Wait Mode) 80
5.3.1. Programación Modo En Espera (Wait Mode).....80
6. Manual de Usuario 82
6.1. Programa inicial.....82
6.2. Terminales finales: PC – GPS. 84
6.2.1. Visualización de datos en tiempo real.....85
6.2.2. Visualización posterior de los datos.....85
6.2.3. Opciones de guardado de los datos.....87
6.2.4. Opciones auxiliares 88
6.3. Terminales finales: PC –PC 89
6.3.1. Funcionamiento en tiempo real 89
6.3.2. Opciones auxiliares 91
7. Presupuesto 93
8. Conclusiones y líneas de futuro 94
8.1. Conclusiones GPS Mode.....94
8.2. Conclusiones User Mode.....95
8.3. Líneas futuras dispositivo.....95
9. Bibliografía.....97
10. ANEXO.....99
10.1. Código programa main.c datalink.....99
10.2. Código programa txsettings.c datalink.....111
10.3. Código programa showtemp.c datalink 112
10.4. Código programa waitMode.m.....117
10.5. Código programa userMode.m.....120
10.6. Código programa TIEMPOREAL.m (GPS Mode).....127
10.7. Código programa abrirRS.m 145
10.8. Código programa cerrarRS.m 145
10.9. Código programa leerRs.m.....145

10.10. Código programa stateRS.m 150
10.11. Código programa userRs.m 150

Índice ilustraciones

Ilustración 1-I Helicóptero PELICANO 5
Ilustración 2-I Pioneer 7
Ilustración 2-II Predator 8
Ilustración 2-III Global Hawk 9
Ilustración 3-I Parámetro S11 (1) 12
Ilustración 3-II Parámetro S11 (2) 13
Ilustración 3-III Pineado CC1111Fx 14
Ilustración 3-IV Display 18
Ilustración 3-V LED's 18
Ilustración 4-I Datalink 21
Ilustración 4-II Esquema microcontrolador 22
Ilustración 4-III Flash Programmer 23
Ilustración 4-IV Esquema pines I/O 24
Ilustración 4-V Diagrama bloques ADC 26
Ilustración 4-VI Protocolo inicial 27
Ilustración 4-VII Display ayuda 27
Ilustración 5-I Modelo de comunicación 30
Ilustración 5-II Diagrama de estados 31
Ilustración 5-III Conjunto GPS 32
Ilustración 5-IV Ejemplo empresa distribuidora 32
Ilustración 5-V Medidores numéricos 39
Ilustración 5-VI Medidores en funcionamiento 40
Ilustración 5-VII PORT COM 40
Ilustración 5-VIII Manual mode 41
Ilustración 5-IX Show Live 41
Ilustración 5-X Live Options 42
Ilustración 5-XI Close COM 42
Ilustración 5-XII Variable a visualizar 43
Ilustración 5-XIII Fast Analize 43
Ilustración 5-XIV Figure Analize 43
Ilustración 5-XV Track 3D 43
Ilustración 5-XVI Archivos de texto 44
Ilustración 5-XVII Funciones workspace 44
Ilustración 5-XVIII Cambio Completo de modo 45
Ilustración 5-XIX Cambio Auxiliar de modo y Solicitud Estado 45
Ilustración 5-XX Track coche (1) 46
Ilustración 5-XXI Track coche (2) 47
Ilustración 5-XXII Mapa ubicación Somosierra 48
Ilustración 5-XXIII Mapa relieve Somosierra 49
Ilustración 5-XXIV Imagen del lugar de vuelo 50
Ilustración 5-XXV Conector antena por defecto 51
Ilustración 5-XXVI Conector antena arreglado 52

Ilustración 5-XXVII Peso equipo 53
Ilustración 5-XXVIII Peso helicóptero 53
Ilustración 5-XXIX Montaje en helicóptero (1) 54
Ilustración 5-XXX Montaje en helicóptero (2) 55
Ilustración 5-XXXI Montaje en helicóptero (3) 55
Ilustración 5-XXXII Montaje en helicóptero (4) 56
Ilustración 5-XXXIII Montaje en helicóptero (5) 57
Ilustración 5-XXXIV Imagen aérea polígono Torrejón 58
Ilustración 5-XXXV Helicóptero preparado para vuelo 58
Ilustración 5-XXXVI Estación terrena para vuelo 59
Ilustración 5-XXXVII Imagen durante el vuelo 60
Ilustración 5-XXXVIII Track visto desde el suelo 60
Ilustración 5-XXXIX Datos altitud 61
Ilustración 5-XL Datos velocidad 61
Ilustración 5-XLI Datos temperatura 62
Ilustración 5-XLII Datos altitud (2) 62
Ilustración 5-XLIII Track con altura 63
Ilustración 5-XLIV Track sin altura 63
Ilustración 5-XLV Track Matlab 64
Ilustración 5-XLVI Track vista altura 64
Ilustración 5-XLVII Track con temperatura 65
Ilustración 5-XLVIII Mapa ubicación Alarilla 66
Ilustración 5-XLIX Despliegue ala 67
Ilustración 5-L Equipo terreno 67
Ilustración 5-LI Equipo aéreo 68
Ilustración 5-LII Salida vuelo 68
Ilustración 5-LIII Datos altura 69
Ilustración 5-LIV Datos velocidad 70
Ilustración 5-LV Datos temperatura 70
Ilustración 5-LVI Track visto desde el lugar del equipo en tierra 71
Ilustración 5-LVII Foto vuelo (1) 71
Ilustración 5-LVIII Foto vuelo (2) 72
Ilustración 5-LIX Escenario real 73
Ilustración 5-LX Opciones COM y modos en User Mode 75
Ilustración 5-LXI Opciones chat User Mode 77
Ilustración 5-LXII Prueba sincronización 78
Ilustración 5-LXIII Prueba conversación 78
Ilustración 5-LXIVOpciones Wait Mode 81
Ilustración 6-I Pantalla Wait Mode 82
Ilustración 6-II Botón GPS Mode 83
Ilustración 6-III Botón User Mode 83
Ilustración 6-IV Botón State 83
Ilustración 6-V Función Puerto COM 83
Ilustración 6-VI Función Cerrar COM 84
Ilustración 6-VII Botonera Modos 84

Ilustración 6-VIII Pantalla GPS Mode 84
Ilustración 6-IX Botón iniciar toma de datos 85
Ilustración 6-X Botonera Tiempo Real 85
Ilustración 6-XI Botón Stop 85
Ilustración 6-XII Opciones Desplegable 86
Ilustración 6-XIII Botón gráfica estándar 86
Ilustración 6-XIV Botón gráfica detalle 86
Ilustración 6-XV Botón gráfica trayectoria 87
Ilustración 6-XVI Botón guardar 87
Ilustración 6-XVII Botón cargar 87
Ilustración 6-XVIII Casilla texto Workspace 87
Ilustración 6-XIX Casilla archivo texto 88
Ilustración 6-XX Casilla archivo GPS 88
Ilustración 6-XXI Función Puerto COM y Tiempo adquisición 88
Ilustración 6-XXII Función Manual Mode 88
Ilustración 6-XXIII Botón User Mode 89
Ilustración 6-XXIV Botón Wait Mode 89
Ilustración 6-XXV Pantalla User Mode 89
Ilustración 6-XXVI Botón Abrir Conexión Serie 90
Ilustración 6-XXVII Casilla texto Chat 90
Ilustración 6-XXVIII Conversación Chat 90
Ilustración 6-XXIX Botón Clear Chat 91
Ilustración 6-XXX Botón Cerrar Conexión Serie 91
Ilustración 6-XXXI Casilla guardar texto Chat 91
Ilustración 6-XXXII Botón Wait Mode 92
Ilustración 6-XXXIII Botón GPS Mode 92

Índice Tablas

Tabla 1 Presupuesto material prototipo (sin gastos de envío) 93
Tabla 2 Presupuesto coste personal 93
Tabla 3 Presupuesto final 93

Agradecimientos

Los agradecimientos de un trabajo de meses, que cierra una etapa de cinco años de universidad, más todos los previos hasta poder llegar aquí, deben ir dedicados a todas y cada una de las personas que a lo largo de mis 23 años han sido capaces de enseñarme algo, sea de la índole que sea, ya que con ese pequeño o gran aporte han tocado algún aspecto de mi personalidad y con ello han influido en mi capacidad de tomar decisiones.

Agradecer en especial a las personas que han puesto empeño en aconsejarme y enseñarme, y no se han rendido aún no encontrando el resultado esperado, porque me han enseñado que con constancia y esfuerzo, se puede conseguir todo lo que te propongas, tan solo hace falta marcarse objetivos.

Dar las gracias a todas las personas que han colaborado en este proyecto, que han servido de mucha ayuda dando un empujón en distintas fases del trabajo, y sin ellas, todo esto se hubiera puesto más cuesta arriba.

Por último agradecer a las personas que me llevan acompañando muchos años, desde mi familia a mis amigos, que son los que realmente me conocen y están ahí para todo, en las buenas y en las malas ocasiones, dándome momentos de alegría que tantas veces ayudan a olvidarse de los problemas que existen, y haciéndome de sentir afortunado por la vida que tengo.

Haciendo mención especial a dos personas que han sido claves en mi vida y ya no están, que hacen que mire cada día como un regalo, como una oportunidad que se tiene que aprovechar porque nunca se sabe si los momentos que dejamos de vivir, algún día volverán, gracias a ellos dos, a mi abuela y a mi amigo Miguel Ángel.

Muchas gracias a todos.

Abstract

In this document studied the system behavior based on chip CC1110 of Texas Instruments, for wireless applications. These devices currently have profusion. Right the increasing demand for control and management wireless applications. In the first part of project presents the state of art of this aspect, with reference to the embedded systems, FPGAs, etc. It also makes a history introduction of UAVs, which are the vehicle for use data link.

In the second part is studied the device through development board, verifying and checking with provided software the scope. The board programming is C language; this gives a good versatility to develop applications. Thus, in third part performing a functionally program, it based on requirements provided by company with which it collaborates, INDRA Company. This program is developed with Matlab, very useful for such applications because of its versatility and ability to use variables.

Finally, with the implementation of such programs, specific tests are performed for each of them, field tests are performed in several cases, and vehicles used for this are the most similar to the actual environment plain to use.

Like implementing with the program made, includes a graphical user manual, so your understanding is conducted quickly and easily.

Ultimately, present future targets for system applications, conclusions, budget and annex of the most important programming codes.

Resumen

Se trata de estudiar el comportamiento de un sistema basado en el chip CC1110 de Texas Instruments, para aplicaciones inalámbricas. Los dispositivos basados en este tipo de chips tienen actualmente gran profusión, dada la demanda cada vez mayor de aplicaciones de gestión y control inalámbrico. Por ello, en la primera parte del proyecto se presenta el estado del arte referente a este aspecto, haciendo mención a los sistemas operativos embebidos, FPGAs, etc. También se realiza una introducción sobre la historia de los aviones no tripulados, que son el vehículo elegido para el uso del enlace de datos.

En una segunda parte se realiza el estudio del dispositivo mediante una placa de desarrollo, verificando y comprobando mediante el software suministrado, el alcance del mismo. Cabe resaltar en este punto que el control con la placa mencionada se debe hacer mediante programación de bajo nivel (lenguaje C), lo que aporta gran versatilidad a las aplicaciones que se pueden desarrollar. Por ello, en una tercera parte se realiza un programa funcional, basado en necesidades aportadas por la empresa con la que se colabora en el proyecto (INDRA). Este programa es realizado sobre el entorno de Matlab, muy útil para este tipo de aplicaciones, dada su versatilidad y gran capacidad de cálculo con variables.

Para terminar, con la realización de dichos programas, se realizan pruebas específicas para cada uno de ellos, realizando pruebas de campo en algunas ocasiones, con vehículos los más similares a los del entorno real en el que se prevé utilizar.

Como implementación al programa realizado, se incluye un manual de usuario con un formato muy gráfico, para que la toma de contacto se realice de una manera rápida y sencilla.

Para terminar, se plantean líneas futuras de aplicación del sistema, conclusiones, presupuesto y un anexo con los códigos de programación más importantes.

1. Presentación del proyecto:

El proyecto persigue como finalidad conseguir un datalink para sistemas no tripulados, por lo que la primera parte del proyecto, coincide con lo que se mostrará en el PFC, conseguir un enlace de datos, que incluyan las medidas realizadas, y el intercambio de dicha información. Y mediante algunos cambios, se conseguirá darle una aplicación final, distinta de la mencionada, que se irá confeccionando según vayan apareciendo las necesidades.

A partir del problema dado, y viendo las características que debe tener, para una aplicación final, nos encontramos con que una solución al problema sería implementar mediante sistemas embebidos un enlace de datos, en el que con medidas de periféricos, un usuario pudiera monitorizar la información que se está adquiriendo en tiempo real, sin necesidad de encontrarse en el lugar de las medidas, y pudiendo cambiar parámetros de la transmisión en el momento que se necesite, mediante una interfaz que se visualizará, gracias a una aplicación conectada al puerto serie RS-232, en el monitor de un ordenador.

El proyecto se realizará mediante una placa de desarrollo de Texas Instruments, Smart RF04EB, y la programación del microcontrolador, que hará las veces de transceptor, CC1110F32 de bajo consumo y a bajas frecuencias.

El entorno de programación que se va a utilizar, es el recomendado por el fabricante, pertenece a la familia 8051 de IAR Systems, siendo el lenguaje de programación C.

Los requisitos del proyecto vienen dados por una empresa, INDRA Sistemas, en la que el autor del PFC, David Rubio Vela, colabora mediante una beca en el proyecto de UAVs (Unmanned aerial vehicle) PELICANO.

Ilustración 1-I Helicóptero PELICANO

El datalink es una de las partes esenciales dentro de un sistema no tripulado, ya que es el encargado de toda la comunicación y el flujo de datos que ello conlleva, siendo algunos de estos, el control en el rumbo, recepción de telemetría o vídeo.

2. Introducción a los UAVs

2.1.Historia

Un vehículo aéreo no tripulado, es aquel que puede volar sin piloto a bordo, controlados por algún tipo de operador a distancia, cumpliendo el resto de requisitos de un vehículo tripulado.

La primera vez que fueron utilizados fue en agosto de 1849, cuando los austriacos atacaron la ciudad de Venecia con globos no tripulados cargados con explosivos. El resultado no fue el esperado, pero sentó un precedente.

Durante la primera guerra mundial empezó la experimentación con “Bombas aéreas no tripuladas”, aunque no fue hasta la segunda guerra mundial cuando se masificaron los aviones no tripulados manejados mediante radiocontrol en los campos de batalla.

Estados Unidos consciente de la importancia de estos sistemas comenzó a utilizarlos en la guerra de Vietnam, aunque su confirmación fue llevada a cabo por Israel durante sus operaciones en el Líbano en 1982.

Algunos de los modelos utilizados por el ejército de los Estados Unidos son el Pioneer israelí, en la operación Tormenta del Desierto en 1991, para suministrar inteligencia a nivel táctico, en la imagen se puede observar la aeronave.

Ilustración 2-I Pioneer

El modelo Predator estuvo presente en la operación Paz Duradera o en el ataque a Irak en 2003, consiguiendo atacar objetivos de gran importancia.

En la actualidad los UAVs como característica principal tienen la de obtención de información útil en tiempo real.

2.2.Componentes de un UAVs

Para cumplir su objetivo debe tener dos partes claramente diferenciadas.

2.2.1. Segmento aéreo:

Compuesto por el vehículo, el conjunto de sensores y los medios para la comunicación y envío de las distintas informaciones, imágenes, etc.

2.2.2. Segmento terrestre:

Compuesto de una estación de control terrestre, estación terminal de datos y todo lo necesario para el lanzamiento, la recuperación y la puesta en marcha del sistema.

Lo anterior refleja la complejidad de la estructura y del personal a cargo, para poder realizar las tareas para las que está diseñado el sistema.

2.3.Clasificación de los UAVs

El término UAVs engloba tanto aeronaves con autonomía de vuelo de hasta 36 horas y alcance de más de 10.000 kilómetros, como a otros pequeños, de poco más de 0.5 Kg y un alcance cercano a los 1.000 metros.

Como no hay mucho consenso en este tema, la siguiente clasificación se basa en la información recogida de la revista del Ejército de Tierra Español número 770 de junio del 2005.

2.3.1. Según la misión:

UAVs Tácticos, satisfacen las necesidades a nivel operativo de las Fuerzas Terrestres con un alcance estimado de unos 200 Km.

UAVs Estratégico/Operacional, de un alcance y altitud superiores, teniendo mayor autonomía, claro ejemplos son el Predator (primera imagen) y el Global Hawk (segunda imagen).

Ilustración 2-II Predator

Ilustración 2-III Global Hawk

2.3.2. Según las tareas a realizar:

- Simulando blancos aéreos, como entrenamiento para las tripulaciones.
- Reconocimiento, dando inteligencia en el campo de batalla.
- Combate, atacando blancos y evitando el riesgo de bajas humanas propias.
- Investigación y desarrollo, mejorando las prestaciones actuales de los existentes.
- Civiles y comerciales, dando exclusivamente este servicio.
- Hay que añadir que los UAVs no solo tienen un cometido en el campo de batalla, sino también como un observador en roles marítimos, control de fronteras o misiones antiterroristas.

2.4.ISR (Inteligencia, vigilancia y reconocimiento)

Este se ha convertido en un factor clave después de la experiencia alcanzada con las misiones realizadas por los UAVs, entre las más importantes se encuentran:

En la guerra de Vietnam, las fotografías realizadas por los UAVs revelaron lugares donde se escondían arsenales de misiles tierra-aire, campos aéreos enemigos, actividades en puertos y evaluación de daños tras la batalla, estas misiones hubieran supuesto poner en riesgo a la tripulación de varios aviones convencionales.

En la operación Tormenta del Desierto contribuyó al éxito táctico de Estados Unidos.

En Afganistán tuvo un papel más ofensivo lanzando 115 misiles e iluminando por láser 525 objetivos.

En la operación Libertad para Irak aún siendo solo el 5% de las misiones de inteligencia, generaron el 50% de la información de blancos fugaces.

A pesar de esto, los UAVs cuentan con un gran número de limitaciones:

No pueden reemplazar plataformas tripuladas capaces de transportar sistemas de mando, control y advertencia.

No son capaces de procesar la misma información que un piloto en la cabina, ni dar una percepción de la situación que lo rodea en los 360º.

La necesidad de mejorar los sensores hace que muchos sistemas que actualmente son utilizados por personal cualificado, no sean capaces de ser totalmente utilitarios.

Aunque a favor de los UAVs hay que decir que se pueden mantener en vuelo mucho más tiempo sin un rumbo fijo, son menos visibles al enemigo, suponen un menor coste y evitan poner en riesgo la vida de los pilotos.

En la comparación entre los UAVs y los satélites, los UAVs salen bastante más económicos, pueden cambiar versátilmente de objetivo y están más cercanos a éste.

Como anteriormente se ha mencionado, los UAVs tienen falta de percepción de la situación, que puede ser subsanada en parte con el apoyo de satélites, aunque es algo que incrementa en gran medida el coste.

Esta integración de distintos sistemas supone un problema en el ancho de banda de la comunicación, pero resolviendo eso, es el camino a seguir, ya que cada vez se exige más a estos sistemas.

Por lo que el avance tecnológico debe de ir acompañando a las Fuerzas Armadas para la consecución y mantenimiento de un sistema UAVs.

3. Características de los sistemas embebidos:

3.1.Definición:

Un sistema embebido, es un sistema electrónico, diseñado para realizar determinadas funciones, normalmente forma parte de un sistema mayor. Su principal característica es el empleo de uno o varios procesadores digitales (CPUs) en forma de microprocesador, microcontrolador o DSP, que aporta 'inteligencia' al sistema del que forma parte.

Para diseñarlo se requieren ingenieros y técnicos, tanto para hardware como software.

3.2.Hardware:

Se encarga del procesamiento de la información generada por sensores o control de determinados actuadores, y su núcleo está formado por una CPU, que puede presentarse en diversos formatos, como son microprocesador, microcontrolador, DSP o FPGA.

Este módulo suele ser desarrollado para cumplir determinados requisitos de la aplicación final, los más reseñables son:

- Tamaño: por lo general suelen ser de tamaño reducido.
- Margen de temperatura: según el ámbito o entorno de aplicación.
- Consumo de energía: se busca minimizarla.
- Robustez mecánica: se debe tener en cuenta por si sufriera golpes o vibraciones durante su utilización.
- Coste: depende de si será un diseño con pocas unidades o de consumo, por lo que se convierte en esencial la calibración de los costes.
- Otros.

En el caso del proyecto se cuenta con un microcontrolador de las siguientes características:

- CPU 8051 de 8 bits.

- 32 KB de memoria flash y 4 KB de memoria RAM.
- Capaz de trabajar en diversos rangos de frecuencia comprendidos entre 300 MHz y 928 MHz.
- Data Rate máximo de 500 kbps.
- Rango de alimentación: 2 a 3,6 V.
- Técnicas de modulación: 2-FSK, GFSK, MSK, OOK, ASK.
- Sensibilidad: -112 dBm.
- Potencia de TX programable: -30 a 10 dBm.
- Tamaño: 6x6 mm.

El coeficiente de reflexión de la antena a la entrada, se ha medido para observar cuál sería una buena frecuencia de trabajo a utilizar:

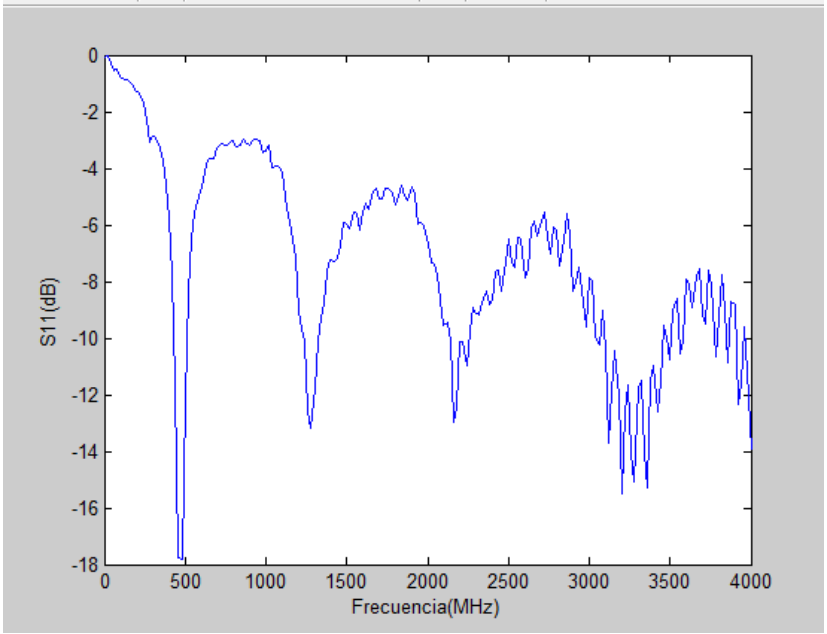

Ilustración 3-I Parámetro S11 (1)

Se observa que por debajo de los 500 MHz es el lugar más propicio para trabajar, llegando practicamente a los -18dB de onda reflejada. Colocando un marker se observa donde se produce ese punto de inflexión:

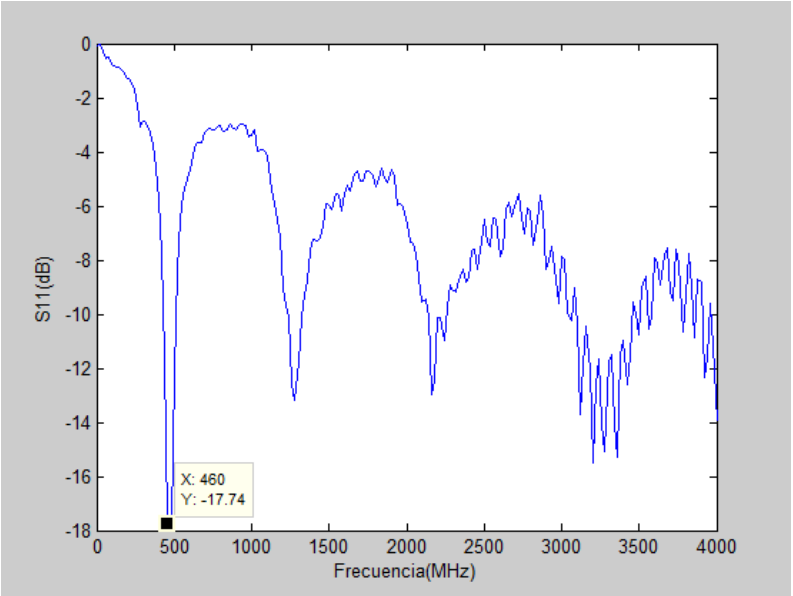

Ilustración 3-II Parámetro S11 (2)

En el entorno de 460 MHz es el mejor lugar donde trabajar con la antena que incluye el equipo, por lo que se buscará una frecuencia cercana de operación.

A continuación se muestra el esquema del sistema con el pineado:

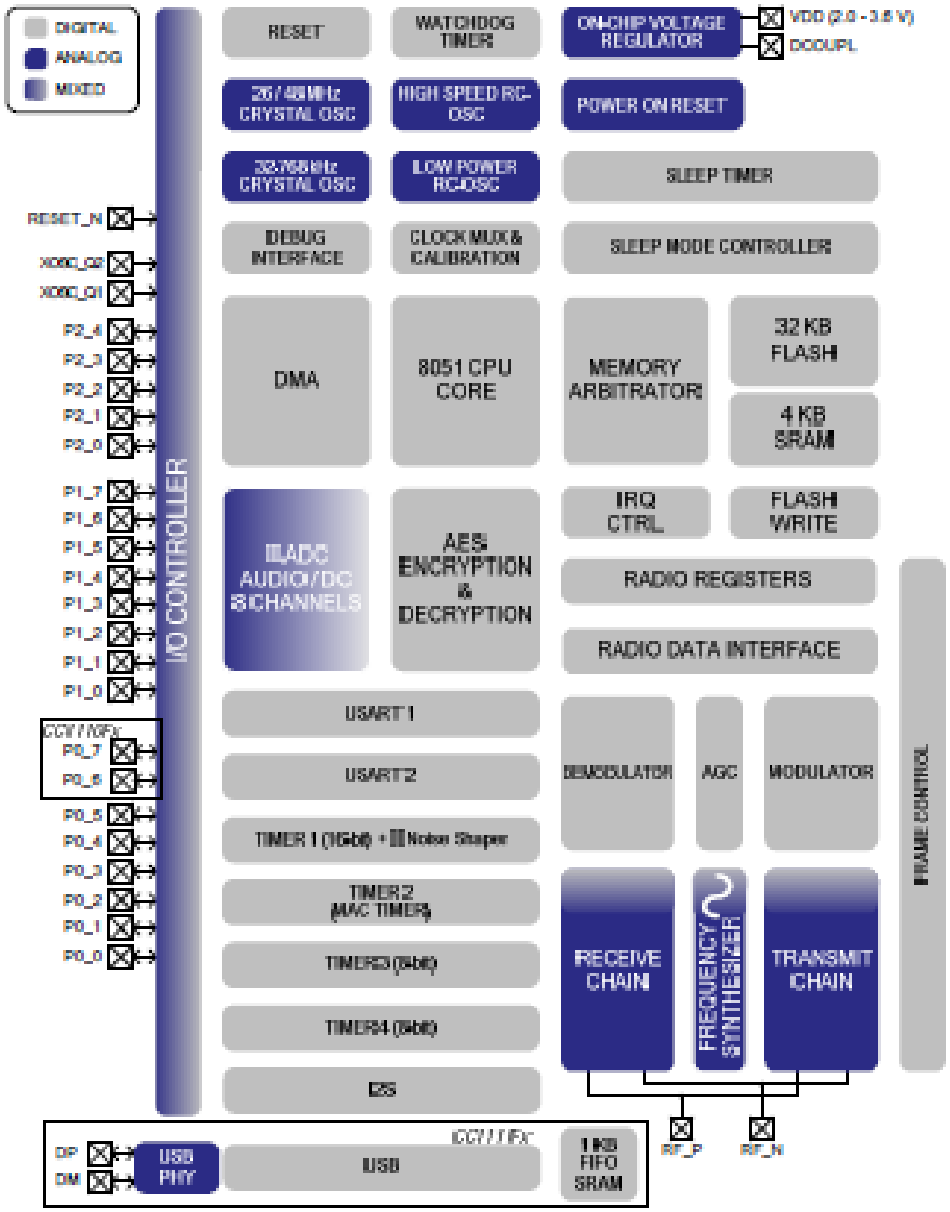

Ilustración 3-III Pineado CC1111Fx

3.3.Software:

El software deberá de cumplir los requisitos impuestos por la aplicación. En el diseño los límites vienen impuestos por la memoria y por la disponibilidad de dispositivos externos. Se tienen las siguientes necesidades:

- Trabajar en tiempo real.

- Optimizar los recursos disponibles.
- Disponer de un entorno de desarrollo específico dependiendo de la familia de microprocesadores utilizados.
- Programación en ensamblador, o de alto nivel, como por ejemplo en C.
- Otros.

El empleo de sistema operativo o no, depende del sistema que se va a desarrollar y del tipo de microcontrolador o DSP a utilizar. Por lo que la decisión se tomará en base a los requisitos del sistema, ya sean técnicos o económicos.

En resumen, un sistema embebido es un sistema basado en un microprocesador cuyo hardware y software son específicamente diseñados y optimizados para poder resolver el problema en concreto, que nos plantea la aplicación final, de una manera eficiente. Su hardware se diseña a nivel de chips, System on Chip, o de tarjeta PCB, en este caso se realiza mediante SoC, y se busca minimizar el tamaño y el coste, maximizando el rendimiento y fiabilidad de la aplicación final.

3.4.Aplicaciones:

Las aplicaciones suelen estar dirigidas a un uso industrial y al mercado de consumo.

En prácticamente todas las áreas de nuestra vida se encuentran sistemas embebidos, un campo a reseñar es el del automóvil, en el que las tareas de seguridad y confort son encargadas a estos sistemas.

Los fabricantes de semiconductores ofrecen su abanico de productos y los relacionan con posibles campos de utilización, éste es el caso del que se va a utilizar en el PFC, Texas Instruments, que los clasifica de la siguiente manera:

- Automóvil y Transporte.
- Electrónica.
- Seguridad.

- Comunicación y Telecomunicación.
- Energía e Iluminación.
- Espacio, Aviónica y Defensa.
- Ordenadores y Periféricos.
- Industria.
- Video e Imagen.
- Medicina.

3.5.Arquitecturas de computadores:

3.5.1. Introducción:

Para un sistema embebido no se necesita de una gran potencia de procesado, ni presentaciones con gran resolución, pero si es requisito indispensable el poder trabajar en tiempo real. Tampoco se contempla la posibilidad de ampliar el hardware con nuevos módulos, ya que el sistema ya está diseñado para cumplir los requisitos marcados, por lo que si se necesitaran modificaciones, esto se realizaría con el rediseño del sistema al completo. Como anteriormente se ha citado, los requisitos más importantes son el tamaño, margen de temperatura, consumo e inmunidad electromagnética.

3.5.2. Micro's:

Como procesador de estos sistemas, y elegido en función de la aplicación, se necesita al menos uno de los siguientes:

Microprocesador: chip que incluye la CPU y la circuitería relacionada con los buses de datos y memoria. Necesita complementarse con chips adicionales, para formar el sistema mínimo, como memoria, circuitos de entrada salida (I/O) y reloj.

Microcontrolador: contiene el sistema mínimo en un chip, por lo que se consta de varias de estas características CPU, buses, reloj, memoria, I/O, conversores A/D, timers.

Procesador Digital de Señal (DSP): son microcontroladores o microprocesadores, que realizan tareas de procesado digital de señales en tiempo real.

El inicio de todos estos dispositivos, se encuentra en el microcontrolador 8051 de Intel, que es la base de muchos de los que aparecieron después, incluso de actuales, ya que sus mejoras hacen que todavía le quede un largo recorrido en esta materia, existiendo una gran oferta de dispositivos.

3.5.3. Comunicaciones:

Los sistemas de comunicación son cada vez mas importantes, y lo normal es que pueda comunicarse mediante interfaces por cable o inalámbricas. Por lo que los sistemas embebidos incorporan puertos bajo los estándares más extendidos, algunos son:

- UART.
- SPI.
- USB.
- Ethernet.
- Fibra óptica.
- Comunicaciones inalámbricas, como pueden ser: WiFi, Bluetooth, GSM, UMTS.

En la placa de pruebas se encuentran los siguientes:

- UART
- SPI
- I^2C

3.5.4. Visualización:

El sistema de TI lleva una pantalla de LCD alfanumérico y diodos LED, que forman parte de la interfaz hombre-máquina, y son de gran ayuda al programador en la realización del software, y al usuario final con la información que estos pueden dar.

Ilustración 3-IV Display

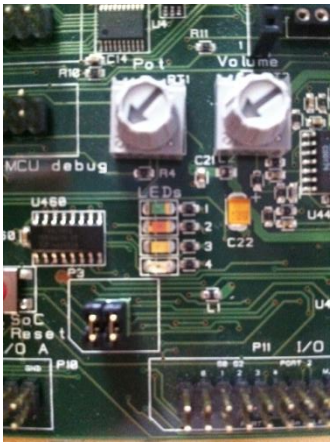

Ilustración 3-V LED's

3.5.5. Actuadores:

Un actuador es un dispositivo capaz de transformar energía en la activación de un proceso automatizado. Recibe la orden de un regulador o controlador y en función de ella activa un determinado elemento de control, como por ejemplo un servoactuador.

En el caso de la placa de pruebas son los encargados de llevar a cabo las acciones que indica la CPU, como drivers de corriente o conmutadores.

3.5.6. Pines I/O analógicos y digitales:

El módulo de Entrada/Salida se encarga de hacer llegar o enviar señales, entre los circuitos encargados de su generación y procesamiento.

La placa cuenta con conversores A/D procedentes de sensores, puertos I/O o encendido y apagado de diodos LED.

3.5.7. Reloj:

Se encarga de la generación de señales de reloj necesarias para la temporización de los circuitos digitales, se parte de un único oscilador, que es parte muy importante en el correcto funcionamiento de las aplicaciones. Los aspectos principales son:

- Frecuencia deseada.
- Estabilidad y precisión de la frecuencia, con las variables de temperatura, envejecimiento…
- Consumo de corriente requerido y su complejidad hardware.
- Coste del resonador que lo compone.

Existen varios tipos de osciladores, basados en resonadores de cristal, cerámicos, SAW, LC o RC.

Cada uno presenta características específicas en valores como el margen de frecuencia, estabilidad y consumo.

El uso de lazos enganchados en fase (PLL) como sintetizadores de frecuencia, permite disponer de un conjunto discreto de frecuencias con gran precisión y estabilidad.

Los dos siguientes, son los que se pueden elegir para el CC1110:

Cristal: mayor estabilidad en frecuencia y menor coste.

RC: menor consumo y tamaño, con gran adaptabilidad a la integración.

Con dispositivos como estos se construyen los Osciladores controlados por Tensión (VCO).

3.5.8. Modulo de alimentación (Power):

Para este tipo de sistemas, es muy importante realizar un dimensionado con las distintas corrientes que se necesitan, tanto en la parte del Master como en el Slave.

En el caso que nos abarca, en la parte Master, la que se encontrará en tierra, no hay ningún problema ya que se dispone de fuentes de alimentación, mientras que para el Slave, colocada en el vehículo de pruebas, se necesitan baterías.

Con los conversores DC/DC se puede partir de un solo valor de tensión y conseguir varios valores, los típicos que se suelen utilizar son 5, 9, 12 y 24 VDC.

Las corrientes de alimentación crean ruido e interferencias que pueden afectar al resto de componentes, por lo que cuando se manejan grandes potencias, es necesario poner apantallamiento en el sistema, se trata de la compatibilidad electromagnética.

En los sistemas embebidos en los que el Slave se encuentra fijo en un lugar de difícil acceso, no es este el caso, la vida del sistema viene limitada por la vida de sus baterías.

3.5.9. Generalidades sobre dispositivos CPU:

La arquitectura de estos dispositivos ha sufrido una gran evolución en los últimos años, buscando la mejora constante.

Hace unos años, diferenciar entre microprocesador y microcontrolador era más sencillo, pero en la actualidad el grado de complejidad que han adquirido los procesos de fabricación, hacen muy difícil separar entre ambos conceptos.

También se han mejorado mucho los DSP, acercando sus funcionalidades a las de un microcontrolador, pero sin perder sus características en el procesamiento digital de señales.

El diseño de un sistema embebido consiste en un modulo PCB, con interconexiones entre sus circuitos integrados y con el resto de componente tanto activos como pasivos.

Un sistema embebido simple, consta de un microprocesador, memoria, algunos periféricos de I/O y un programa que se dedica a la realización de una aplicación concreta.

4. Transcurso del proyecto:

El comienzo del proyecto viene a partir del ofrecimiento en la empresa INDRA, mediante la persona de Andrés España Fresno, ingeniero de Telecomunicación y tutor del autor del PFC en la empresa, del desarrollo de un datalink para ser utilizado en UAVs's, para llegar a ver la capacidad de la que dispone el dispositivo.

Con esta idea, el profesor de la EUITT, Francisco José Arqués Orobón, estuvo de acuerdo con el proyecto, y añadió que al datalink se debería de buscar alguna aplicación final, y se decidió utilizar algunos sensores que utilizaran entradas distintas, y mirando las posibilidades disponibles y que fueran acordes con un enlace de datos real utilizado en UAVs, se decidió coger un GPS que mediante un conector RS232 diera su posición al terminal en tierra en tiempo real, y mirando las opciones del propio datalink, se decidió que el sensor analógico de temperatura interno tras pasar por el ADC diera a conocer su valor, también se podría haber escogido cualquier sensor externo como un sensor de la presión barométrica o un acelerómetro.

La placa de desarrollo que proporciona INDRA, de Texas Instruments, y el microprocesador CC1110F32, será el punto de inicio del proyecto.

Ilustración 4-I Datalink

4.1.Programación básica en C del datalink:

Mediante los programas que se proporcionan en la web de Texas Instruments, comienza la primera parte del proyecto.

Sin el archivo auxiliar con las etiquetas de las direcciones de memoria sería un trabajo muy laborioso para poder arrancar con el proyecto, ya que se tendría que etiquetar con su respectiva posición de memoria todos los registros que deseáramos utilizar, pero por suerte viene incluido.

Con el estudio programas sencillos facilitados en la web de TI, y con el apoyo de algunas de las funciones que estos realizan se ve como poder programarlo.

El programa principal a utilizar como base es uno que a partir de la realización del envío de un determinado número de paquetes, con las opciones seleccionables de la frecuencia y el data rate, nos muestra el número de paquetes que se han recibido. Aunque hay funciones que son de gran ayuda como pueden ser el UART o los pines I/O.

Tras compilar el programa, se carga en la placa a través de un conector USB, por el que también se alimentará la placa durante toda la fase de perfeccionamiento del programa y pruebas de nuevas funciones.

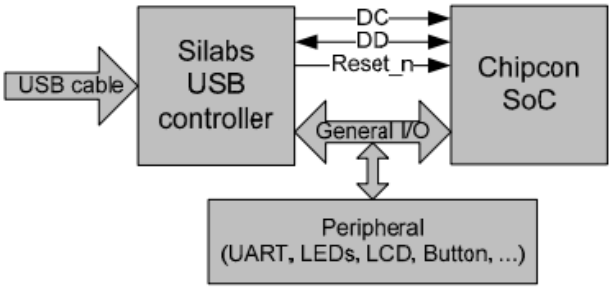

Ilustración 4-II Esquema microcontrolador

El programa que se utiliza para cargar el código en el dispositivo está disponible en la página web de Texas Instruments, este es el SmartRF Flash Programmer.

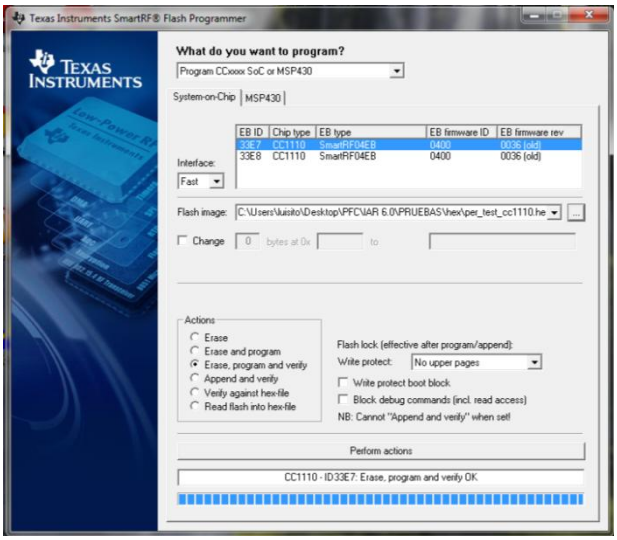

Ilustración 4-III Flash Programmer

El proyecto se empezará mediante funciones triviales e irá aumentando en complejidad.

Los primeros pasos son los más sencillos, el primero de todos es hacerse con el control del display disponible para que nos sirva de ayuda durante la nueva carga de un programa, mostrando por pantalla frases indicando en que parte del programa se encuentra, para cuando se produzca un error en el funcionamiento se pueda encontrar con más facilidad.

Otro de los primeros pasos es el del encendido y apagado de LED, que será utilizado en el futuro para indicar si se encuentra transmitiendo o recibiendo. Esto será de gran ayuda para ver la sincronización de los dispositivos.

En la siguiente tabla se muestra como esta realizado el pineado con la placa, para poder por ejemplo encender los LED que se han comentado, esto se puede realizar mediante software, con un código, o mediante hardware, aplicando en el pin una tensión, que le haga pasar a su estado activo.

Pin no.	Pin name (USB MCU)	Signal name SmartRF®04EB	0-ohm resistor	Function
1	P0.1	P1.7/SO/GDO1/MISO	R117	SPI MISO signal, transceiver/transmitter SO/GDO2
2	P0.0	P1.5/SCLK	R115	SPI Serial clock
10	P3.0/C2D			USB MCU Debug pin
11	P2.7	P1.3/LED3	R113	LED3 (yellow), active low
12	P2.6	P0.4/RTS	R100	
13	P2.5	RS232_POWER		Turns RS-232 level converter on/off
14	P2.4	P2.0/LED_4	R120	LED4 (Blue), active low
15	P2.3	RESET_N		LCD Power on reset signal, SoC RESET
16	P2.2	SOC_PRESENT		Tells USB MCU whether a SoC is present. 0 = transmitter/transceiver 1 = SoC
17	P2.1	P0.6/JOY	R106	Joystick input (analogue coded voltage)
18	P2.0	P1.2/LED2	R111	LED2 (Red), active low
19	P1.7	P1.0/LED1	R110	LED1, (Green), active low
20	P1.6	P0.7/POT	R107	Potentiometer input
21	P1.5	P0.5/JOY_PUSH	R112	Joystick pushed
22	P1.4	P1.1/PWM_OUTPUT	R105	PWM audio output
23	P1.3	P0.1/BUTTON_PUSH	R101	Button pushed
24	P1.2	P0.0/MIC_INPUT	R104	Audio input
25	P1.1	P2.4/SCL	R124	I2S clock (for LCD)
26	P1.0	P2.3/SDA	R123	I2S data (for LCD)
27	P0.7	P2.2/GDO2/DC	R122	Transceiver/transmitter GDO3, SoC debug signal
28	P0.6/CNVSTR	P2.1/GDO1/DD	R121	Transceiver/transmitter GDO1, SoC debug signal
29	P0.5	P0.2/UART_RD	R102	UART RD
30	P0.4	P0.3/UART_TD	R103	UART TD
31	P0.3/XTAL2	P1.4/CSn/SS	R114	SPI slave select signal
32	P0.2/XTAL1	P1.6/MOSI	R116	SPI MOSI signal, Transceiver/Transmitter SI

Ilustración 4-IV Esquema pines I/O

Se irá aumentando la complejidad, el siguiente paso es conseguir un enlace de datos, en el que se manda un paquete con un número, que irá incrementando en cada envío y mostrando en el display del receptor.

Esta transmisión también se irá modificando, primeramente se realizará solamente en una dirección, para ver que se transmite, mas tarde mediante un “testigo”, que se entrega entre los transceptores, y en el que se realiza un envío y se espera la respuesta.

Para poder realizarse el intercambio de información, se han tenido que seleccionar varios parámetros, como la frecuencia a 433MHz, el data rate a 500Kbaudios, la potencia de salida a 10dBm y un oscilador a cristal para el modo de alto rendimiento.

Los parámetros que varían estos datos vienen incluidos en los archivos ‘txsettings’.

Sin olvidar que para la transmisión y recepción, ya debe de estar implementada una interrupción, y ésta será en forma de vector, lo que hará que el código vaya a atender la interrupción en el momento en el que se produzca, y al terminar esta, el código seguirá ejecutándose desde el punto en el que fue interrumpido.

Las interrupciones se programan en la función principal, al final de ella, con la sentencia #pragma dando a la variable “vector” el valor de la interrupción que se produce por radiofrecuencia.

Cada mensaje que se envía debe de tener en sus primeros bytes la longitud del paquete y un código de identificación de la red, para que cuando lo reciba el otro par y lea esos valores sepa el tamaño de bytes a leer, y si el paquete es para él, sino lo fuera se descartaría.

```
radioPktBuffer[0] = PKTLEN;                      // Packet length
radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);  // Network
radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;       // identifier
```

Los paquetes tienen definido un tamaño, al que ahora mismo no se presta mucha atención, ya que aún no se sabe qué tipo de datos va a tener que transportar, por lo que depende del punto en el que el desarrollo se encuentre, cuando se tengan todos los datos a transportar definidos, entonces se optimizará el paquete.

Todas las interrupciones que se realicen en el programa funcionarán de manera similar, y habrá que dar prioridades para cuando coincidan varias, se atiendan por el orden que se ha estimado sea el más correcto.

El siguiente paso en el intercambio de información, es el de medir la temperatura del sensor interno de la placa de desarrollo, y enviar la temperatura al otro dispositivo, mostrando por pantalla ambas medidas (la temperatura propia y la temperatura del otro dispositivo).

Estas sentencias vienen reflejadas en los archivos ‘showTemp’.

Estas acciones llevaron mucho tiempo para poder realizarse, porque es la primera en la que se deben tratar datos, ya que venían estos en complemento a dos y debían pasarse por el ADC, para poder ser mostrados por el display.

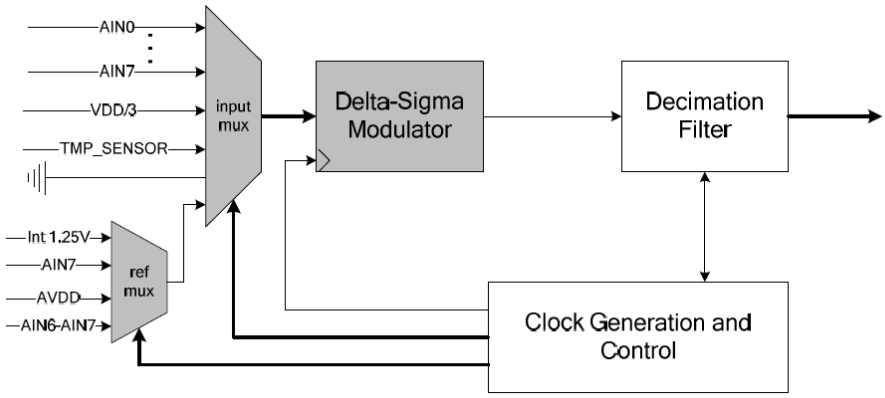

Ilustración 4-V Diagrama bloques ADC

Para realizar la conversión no se utiliza una interrupción, se consigue el dato y se convierte cada vez que se crea un paquete, para después enviarlo.

Llegado a este punto, ya se tiene un sistema de comunicación, bastante básico, en el que se envía un paquete y se espera a que llegue una respuesta para poder volver a envíar, si esto no sucede se acabo la comunicación porque los dos dispositivos estarían esperando en modo receptor de manera infinita.

Antes de solucionar ese problema, se va a crear un sistema de envío, basado en una comunicación TDMA, antes se ha hecho uno con el envío de diez paquetes y espera de otras tantas respuestas con similares resultados, en el que en principio el tiempo de envío será al 50%.

Para esto se debe programar un timer, elegimos el Timer2, que cuenta con 8 bits y un preescaler, y hará que durante su proceso de cuenta se permanezca en el estado actual, y al finalizar se cambie.

Para poder llevar a cabo este protocolo de comunicación debe de existir sincronización, algo que se consigue mediante el envío cada cierto número de paquetes con el estado actual del timer, para que su par lo actualice. También existen unos tiempos de guarda, para evitar la pérdida de paquetes. Por lo que gráficamente queda de la siguiente manera:

TX		RX		TX
RX		TX		RX

Ilustración 4-VI Protocolo inicial

Antes de que comience la transmisión de uno de los pares, el otro ya está en el estado de recepción, y antes de que termine la recepción, el transmisor ha acabado su tarea, estos tiempos no se pueden percibir a simple vista, ya que son del orden de milisegundos.

En este punto ya contamos con un enlace de datos sincronizado, en el que el tiempo de envío está dividido entre los dos transceptores de igual manera.

Esto es algo que más tarde se variará, ya que para la aplicación final se transportan muchos más datos en una dirección que en la otra.

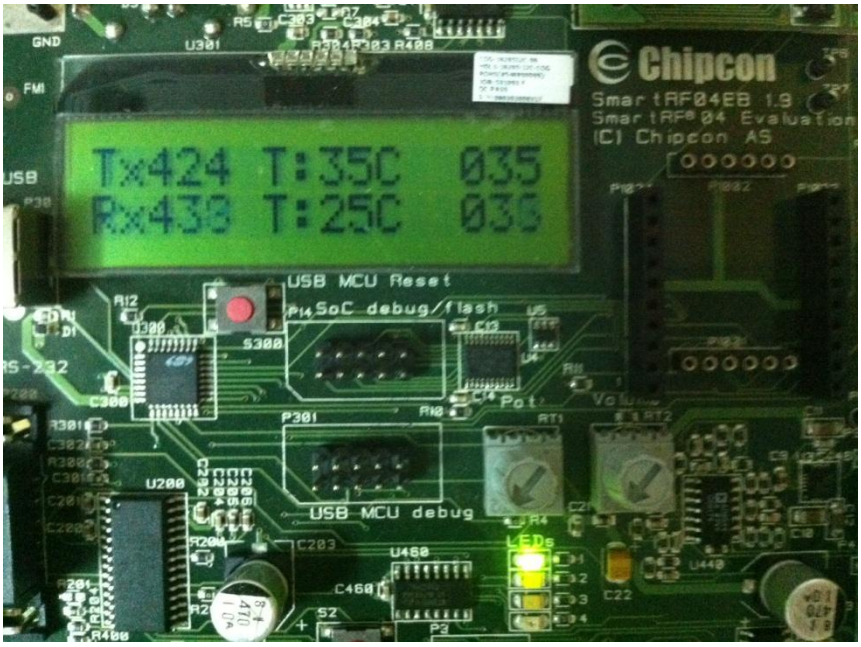

Ilustración 4-VII Display ayuda

En la imagen se puede ver los resultados de las transmisiones, que cuenta con tres tipos de datos:

- En la primera columna, están presentes el número de paquetes enviados y recibidos totales.
- En la segunda, la temperatura propia y la del otro dispositivo (la variación es debido a que están sin calibrar los sensores, se debería sumar o restar un valor para la calibración).
- En la tercera y última columna, se ve el número de paquetes enviados y recibidos en la última transmisión o recepción, según corresponda.

El siguiente paso, es poder controlar el enlace de datos, cambiando sus parámetros, como el data rate, el tamaño de paquete, o los modos de transmisión-recepción (siendo bidireccional o unidireccional en cualquiera de los dos sentidos).

Esto se conseguirá a través del puerto serie RS-232, que será la parte que permita al usuario controlar la comunicación y los cambios de parámetros, esto se hará a través de hyperterminal, en un primer momento, que es una interfaz de fácil manejo, en la que se verá lo que está ocurriendo en la comunicación.

Para realizar esta comunicación, se debe habilitar la interrupción del USART0 del dispositivo, que es el que se ha elegido entre los dos disponibles, da la opción de trabajar de manera asíncrona (UART) o síncrono (SPI), se elige el modo asíncrono, por lo que queda que el que se utilizará será el UART0, con una configuración de las siguientes características:

- 8 bits de datos.
- Bit de comienzo a bajo nivel y de finalización en alto.
- Paridad desactivada.
- Un bit de parada.
- Baudrate de 9600bps.

En primera instancia, se va a trabajar con estos valores, debido a que no se tiene que adaptar a ningún dispositivo externo. Por lo que son valores provisionales.

La herramienta del hyperterminal será de utilidad, aparte de para visualizar en tiempo real el estado de la comunicación y modificarla, para mostrar los datos que nos proporciona el datalink, ya que los suministra en formato texto.

Para guardar los datos e intentar no perder ninguno, se pensó y se implemento una cola dinámica, que mas tarde fue descartada por el retardo que producía en el procesado, y fue

cambiada por un array de valores, que al llegar al último volvía a sobrescribir sobre el primer valor.

Esto es debido a que mediante el puerto serie puede mandarse unos datos al dispositivo para enviar a su par, y el micro encontrarse recibiendo, por lo que los almacenaría de esta forma, y cuando llegara el momento de enviar sacaría estos datos a la transmisión según el orden de llegada.

En este punto se ha conseguido un protocolo de comunicación, con la medida del sensor de temperatura interno, y la posibilidad de cambiar sus protocolos y opciones de transmisión. Por lo que se tiene una comunicación genérica, en este momento se debe decidir hacia donde encaminar el proyecto para conseguir la aplicación final.

5. Separación datalink genérico con la funcionalidad específica:

Llegado a este punto se realizan unas aplicaciones finales para estos dispositivos, algo que sea visual y de fácil manejo, que complemente el trabajo realizado para poder mejorar el producto, estas aplicaciones se explican y muestran en los siguientes apartados.

En esta figura se puede observar la torre del modelo de comunicación que se obtiene con la totalidad del equipamiento.

Existen tres modos de operación, GPS Mode, User Mode y Wait Mode.

En GPS Mode el camino total comprenderá una torre lateral con la central, siendo los datos GPS tratados en el código Matlab y enseñados al usuario final mediante la aplicación creada, mientras que en User Mode la comunicación llegará entre los usuarios extremos, pero en la torre central, tan solo ascenderá hasta el nivel código C, en el que el paquete debe ser reenviado a su destino final.

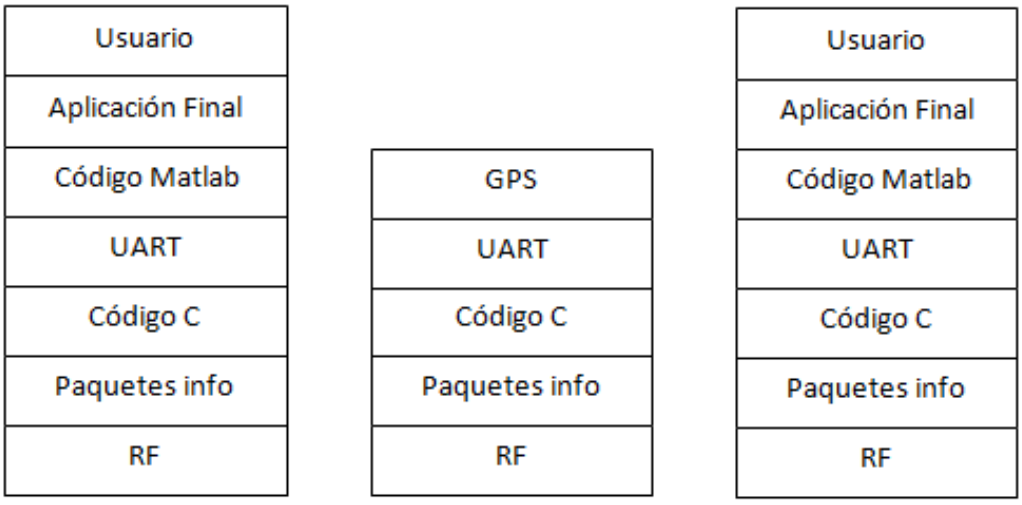

Ilustración 5-I Modelo de comunicación

Y el diagrama de estados entre sus modos, deja moverse entre cada modo directamente, y el diagrama sería el siguiente:

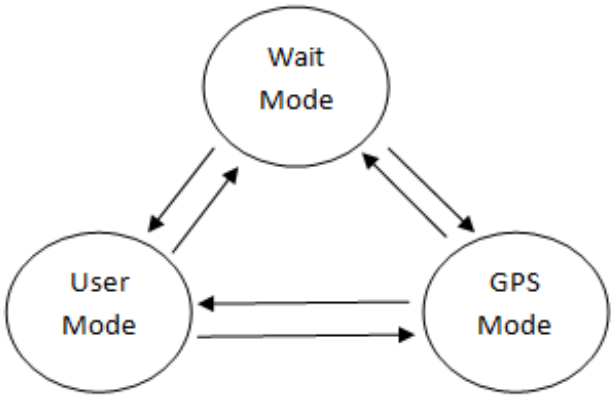

Ilustración 5-II Diagrama de estados

5.1.Modo Localización(Location o GPS Mode)

Tras conseguir el correcto funcionamiento del flujo de datos, y mediante la ayuda del tutor del proyecto, Francisco José Arqués, se procede a la búsqueda de un módulo de GPS, para poder acoplarlo en el datalink.

Esta búsqueda se realiza a través de internet, dentro los equipos convenientemente documentados, y realizando una comparativa de prestaciones adaptadas a las necesidades del proyecto, se decide comprar en el distribuidor SparkFun, un receptor de GPS de 20 canales EM-406A SiRF III con la antena incluida, y una placa de evaluación GPS, que hace la función de interconexión entre el receptor y la placa de desarrollo del datalink a través de un cable RS-232.

Ilustración 5-III Conjunto GPS

Se decide la compra de este GPS, ya que dentro de la gama que ofrece Sparkfun, es uno de nivel medio y no tiene un precio muy alto.

Muestra de ello es un experimento que recogen en su web entre los distintos receptores GPS.

El que se ha comprado es el que aparece con la trama verde, la prueba completa viene registrada en http://www.sparkfun.com/tutorials/169

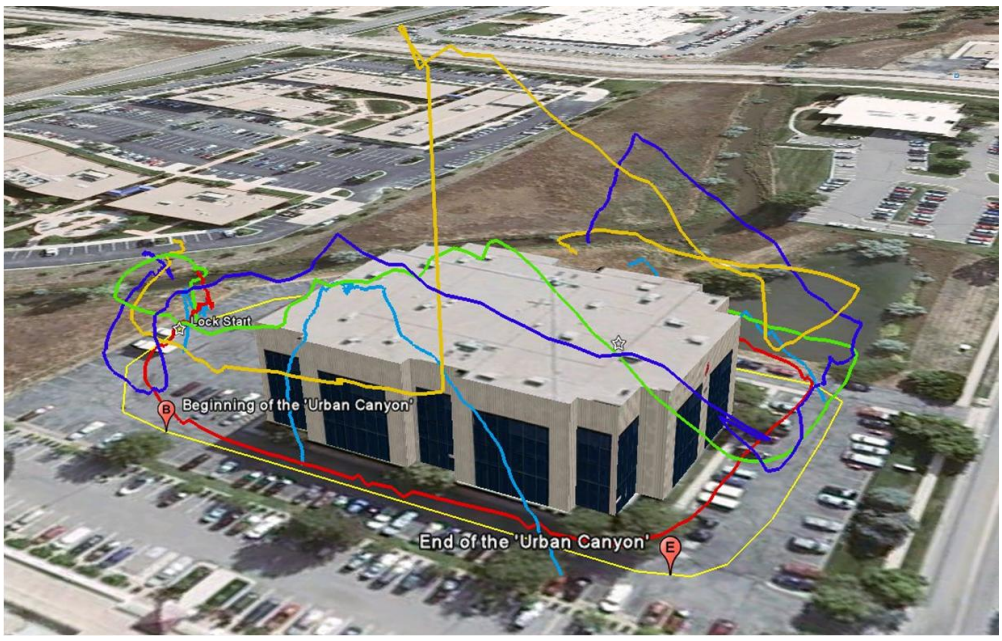

Ilustración 5-IV Ejemplo empresa distribuidora

La idea final para el proyecto, es que un extremo de la comunicación se ocupe de medir la posición GPS acompañado de la temperatura, lo mande mediante paquetes al otro dispositivo, que lo entregará al ordenador de una manera estructurada para que sea fácil poder tratar con los datos.

En primer lugar, como medida de prueba, para verificar el funcionamiento, se realizará la prueba dentro del lugar de la programación, para comprobar que la comunicación es correcta, observando las tramas mediante hyperterminal.

5.1.1. Características principales receptor GPS:

Frecuencia: 1575,42 MHz

Tasa de chip: 1.023 MHz

Sensibilidad: -159 dBm

Precisión posición: 5-10 metros.

Precisión velocidad: 0,1 m/s.

Precisión tiempo: 1 µs sincronizado con la hora de GPS.

Datos en WGS-84.

Tiempo de adquisición medio: 0,1 segundo.

Tiempo de arranque medio: 1 segundo.

Tiempo de calentamiento en el arranque: 38 segundos.

Tiempo de arranque en frío: 42 segundos.

Altura máxima: 18.000 metros.

Velocidad máxima: 515 metros.

Aceleración máxima: 4G.

Alimentación y consumo: 4,5 – 6,5 V DC y 44 mA

Voltaje de salida: 0 – 0,285 V

Baud rate: 4.800 bps

Mensajes de salida: NMEA 0183 GGA, GSA, GSV, RMC, VTG, GLL

Dimensiones: 30mm*30mm*10,5mm ±0,2mm

Temperatura de operación: -40ºC – 80ºC

5.1.2. Programación Modo Localización (Location o GPS Mode):

Lo primero para empezar a programar la implementación con el datalink, es conocer los datos que nos va a proporcionar el GPS mediante el protocolo NMEA.

NMEA: National Marine Electronics Association, esta asociación desarrolló un tipo de especificaciones para que los sistemas electrónicos se pudieran interconectar entre ellos. Con este standard se logró el envío y recepción de datos entre los distintos equipos.

La comunicación mediante GPS, esta implementada mediante este protocolo, y está ideada para que sea autosuficiente para la recepción de tres datos, PVT (posición, velocidad, tiempo).

Cada línea de sentencia comienza con '$' y continua con un prefijo dependiente de la aplicación, en este caso es GP, completando la información otras tres letras, y terminando con un retorno de carro. La línea de mensaje no puede exceder los 80 caracteres.

Los datos contenidos dentro de cada línea van separados por comas, y varía su longitud en función de la precisión con la que se esté dando el mensaje. Terminando el mensaje con una comprobación, que es '*' y dos dígitos en hexadecimal.

En el código del dispositivo, en la parte de la interrupción, para habilitar el modo localización y deshabilitar el que este activo en ese instante, se debe enviar la secuencia '&l', y con esto se habilitan dos variables que son para cambio de modo (en caso de conflicto entre los pares de

dispositivos, prevalece el que tenga esa variable activada) y otra para habilitar el modo GPS de receptor.

changeM = TRUE;

CGPSM = TRUE;

Con la segunda variable, el dispositivo que recibe la información, envía a su par un paquete en el que le pide que se cambie al modo GPS, para que el dispositivo remoto sea el que envíe las tramas que lee a través de la conexión RS232, y el dispositivo local las recibirá y las enviará por RS232 al ordenador. Al funcionar el baud rate del GPS en 4.800 bps, se tiene que cambiar la configuración del dispositivo encargado de recibir los datos del GPS a esa tasa.

En esa misma interrupción más adelante se encuentra la parte en la que el dispositivo remoto, que tiene habilitado el modo GPS de emisor (GPSM = TRUE), en la que al leer el carácter '$' inicializa la cadena de caracteres en la que almacena y lo guarda en la primera posición, en las siguientes lecturas sigue almacenando por orden de llegada, preguntando siempre si el carácter leído es '\n' que es final de trama, para acabar la cadena de caracteres y almacenarla en el array si es una de las que se ha considerado útiles de las varias que tiene NMEA, y casualmente coincide que las útiles son las que en el cuarto carácter no tienen la letra 'S', por lo que es fácil descartar las otras. Posterior a esto se rellena con caracteres no válidos el vector, hasta el valor 80 que es el máximo que tiene definido el protocolo, para evitar errores.

El código que realiza esta tarea es el siguiente:

```
//Trama GPS
else if (GPSM == TRUE)
{
if(U0DBUF == '$'){
      uartRxIndex=0;
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;
}
else{
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;
}

//Final de trama
if(U0DBUF == '\n'){
```

```
        if (AUXleerRS[4]!='S')
        {
                strcpy(NMEAdata[jG].data,AUXleerRS);
                NMEAdata[jG].active=1;
                jG++;
                uartRxIndex=0;
                if(jG==10){jG=0;}
        }
        do{
                AUXleerRS[uartRxIndex]='\0';
                uartRxIndex++;
        }while(uartRxIndex<80);
        uartRxIndex=0;
        }
}
```

El dispositivo en recepción, está en el modo RGPSM, y al recibir un paquete y comprobar que es correcto, lo envía por RS232 al ordenador.

Para poder cambiar a otro modo de funcionamiento, los dispositivos deben sincronizarse para que al realizar el cambio transmisión-recepción, se realice de una forma correcta.

Aquí entra el problema de la tasa de transmisión, con la que exige el receptor GPS, podría elegirse entre dos opciones:

- Trabajar con el dispositivo encargado de recibir del GPS con una tasa de 4.800 bps y el encargado de llevar los datos al ordenador a 9.600 bps.
- Trabajar con ambos dispositivos a 4.800 bps de baud rate.

Se decide elegir la segunda, mirando los pros y los contras.

Lo que aporta la segunda es reciprocidad entre los dispositivos, ya que ambos pueden utilizarse en ambos sitios de la comunicación, la programación del código es idéntica, tanto en C como en Matlab, y deja abierta la puerta a otras aplicaciones, que sean compatibles con ambos dispositivos.

En contra tiene que con 4.800 bps, el dispositivo que se encarga de recibir por RF la información GPS y enviársela al ordenador, se ve muy apurado de tiempo, para llegar a recibir los siguientes datos.

Para evitar que esa pérdida de información sea muy habitual, la sincronización de los dispositivos se realiza cada 10 segundos, en esa sincronización existe un riesgo mucho mayor de pérdida de información. Pero al ser poco habitual y la pérdida de información poco crítica, se decide por elegir esa opción.

En este punto entra en funcionamiento otro tipo de programa, aún no descrito.

Al encontrarse la necesidad de recibir esos datos en un ordenador a tiempo real, se contemplan las diferentes opciones que podrían ser válidas, y se decide por escoger el entorno de programación Matlab como base para esta funcionalidad.

Una de las funcionalidades de Matlab es su GUI, Guide User Interface, la cual parece que puede cubrir las necesidades que se presentan.

La toma de contacto con GUI es rápida, debido a que hay gran cantidad de tutoriales en distintas páginas webs, incluyendo los propios de Matlab, que son de gran utilidad para casos muy concretos.

Con sencillos pasos se puede empezar a programar, las primeras pruebas comprenden la utilización de botoneras que activan funciones, mensajes de texto explicativos en la interfaz, utilización de gráficas, presentación con imágenes, etc.

Tras esto y haber adquirido una manejabilidad aceptable con el entorno, se procede a probar la lectura de textos que reciben a través del puerto serie, para ello tras la configuración pertinente del puerto, se lee un número determinado de caracteres que se mostrará en pantalla, tras la lectura se cierra el puerto para evitar problemas a la hora de abrirlo de nuevo.

Con el manejo de la lectura de datos, se debe de implementar una manera de leer los datos sin dar posibilidad a las confusiones, para ello observando las tramas NMEA, se decide que las claves para la lectura serán el carácter inicial de las tramas ‘$’ y las comas ‘,’ que separan las distintas variables.

Por lo que por ejemplo para la lectura de altitud en la trama $GPGGA, tras leer ‘$’ y reconocer que es la trama ‘GPGGA’ se leerán 9 ‘,’ y a continuación el dato numérico hasta volver a tener la lectura de otra ‘,’.

Lógicamente la lectura de cada dato que se necesita, se realizará por orden de colocación en la trama y se almacenará en una variable que alberga únicamente ese valor, que al terminar de leer la secuencia completa de tramas correspondientes a ese momento de recepción,

almacena su valor como el último en una variable array que se ocupa de almacenar el valor de ese tipo, en toda la comunicación.

Por el orden de aparición de las variables subrayadas se encuentra, en $GPGGA: hora, latitud, longitud y altitud, y en la trama $GPRMC: velocidad y fecha.

```
$GPGGA,001430.003,3907.3885,N,12102.4767,W,1,05,02.1,00545.6,M,-
26.0,M,,*5F
$GPGSA,A,3,15,18,14,,,31,,,23,,,,04.5,02.1,04.0*0F
$GPGSV,3,1,10,15,48,123,35,18,36,064,36,14,77,186,39,03,36,239,29*7A
$GPGSV,3,2,10,09,08,059,,31,35,276,35,17,10,125,,11,08,306,*79
$GPGSV,3,3,10,23,41,059,37,25,06,173,*70
$GPRMC,001430.003,A,3907.3885,N,12102.4767,W,000.0,175.3,220403,015.4,
E*71
$GPGGA,001431.003,3907.3885,N,12102.4767,W,1,05,02.1,00545.5,M,-
26.0,M,,*5D
```

Mediante la programación realizada en los dispositivos, entre la trama $GPRMC y $GPGGA aparece la de la temperatura, que se eligió ese lugar porque es en el que el receptor GPS esta varias decimas de segundo sin recibir información, y comienza con el identificador /TEMP.

```
/TEMP,24.7,C
```

Por lo que una secuencia final sería de la siguiente manera:

```
$GPGGA,112151,4026.664,N,330.0967,W,1,05,02.1,574.3,M,-26.0,M,,*5F
$GPGSA,A,3,15,18,14,,,31,,,23,,,,04.5,02.1,04.0*0F
$GPGSV,3,1,10,15,48,123,35,18,36,064,36,14,77,186,39,03,36,239,29*7A
$GPGSV,3,2,10,09,08,059,,31,35,276,35,17,10,125,,11,08,306,*79
$GPGSV,3,3,10,23,41,059,37,25,06,173,*70
$GPRMC,001430.003,A,3907.3885,N,12102.4767,W,0.03,175.3,120712,015.4,E
*71
/TEMP,28.2,C
```

Aunque como se ha comentado anteriormente las tramas con el carácter ‘S’ en cuarto lugar después del inicial ‘$’, son descartadas en el dispositivo emisor, por lo que no llegan a ser tratadas por el ordenador en tierra.

Que a través de una funcionalidad del programa, saca un tipo de trama creada, con todos los datos que se han pensado útiles, por orden de aparición las variables son hora, latitud,

longitud, altura, velocidad (km/h y nudos), temperatura y fecha, además de añadir un identificador. La trama es la siguiente:

```
/GPSTEMP,112151,4026.664,N,330.0967,W,574.3,M,0.05556,KM/H,0.03,K,28.2
,C,120712,ID7
```

Los valores importantes que se tomarán y representarán gráficamente en tiempo real son altitud, velocidad (km/h) y temperatura, valores que aportan gran información y son fácilmente representables, para ayudar a la lectura de los datos cuando se produzcan comunicaciones muy largas habrá un par de gráficas de cada valor leído, uno con el historial completo de toda la comunicación producida, y otra con los últimos 10 segundos, para conseguir una nitidez mayor en los valores y las evoluciones de estos.

También se han incorporado unos medidores de todos los valores importantes en tiempo real, que representan en la propia interfaz principal del GPS Mode, los valores numéricos de velocidad, tanto en km/h como en nudos, altura, temperatura, latitud y longitud.

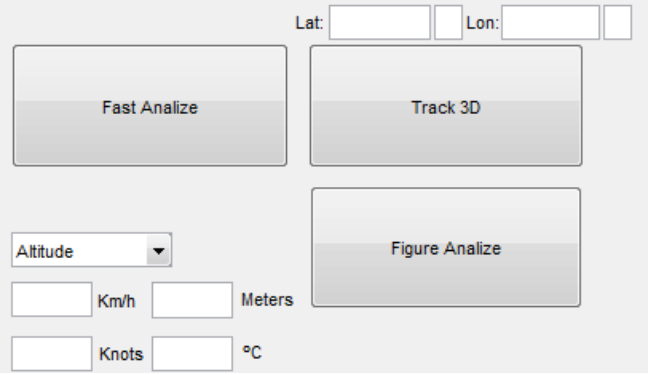

Ilustración 5-V Medidores numéricos

Las casillas donde se representan estos valores se encuentran en la parte superior izquierda y vienen acompañados de sus unidades.

En la siguiente imagen se observa los displays en funcionamiento en un determinado momento de la comunicación.

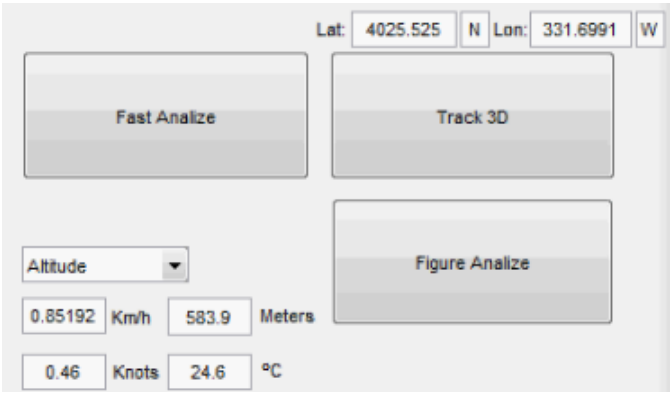

Ilustración 5-VI Medidores en funcionamiento

Para realizar la comunicación lo primero es decir al programa de qué puerto serie tiene que leer los datos, esto se hace eligiendo el número en la casilla PORT COM y validando al pulsar OK.

Cuando se pulsa OK, se leen los valores de las casillas del puerto y los segundos, y se almacenan en unas variables, el puerto en una variable global, debido a que tiene que utilizarse en otro fichero .m y es la mejor manera de que el valor se transfiera, y los segundos con el tipo de variable Matlab "handles" que son variables locales utilizadas en el programa siempre que se encuentren en el mismo fichero.

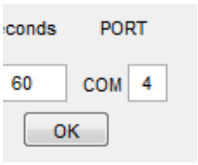

Ilustración 5-VII PORT COM

El tiempo de adquisición de datos se puede definir de dos maneras, dando un tiempo fijo antes de iniciar la comunicación y validando pulsando OK (no es necesario esperar el tiempo programado, se puede parar mediante Stop), o teniendo seleccionada la opción Manual mode, que deshabilita la anterior y estará en comunicación hasta que se pulse Stop.

Ambas variables son de tipo global, esto es debido a que al trabajar en tiempo real en el programa de adquisición de datos, solo es posible cambiar los valores de estas con este tipo de variables, ya que por ejemplo las tipo handles, necesitarían de finalizar las funciones para guardar su valor y que pudiera ser utilizado por otra.

El funcionamiento de los modos es sencillo, en Manual mode, lo que se hace es tener el valor de los segundos totales de conexión un valor por encima de los segundos transcurridos, así nunca llegará a finalizar, mientras que pulsando Stop, lo que sucede es que los segundos transcurridos valen una unidad más que los segundos totales, por lo que finaliza el bucle que mantiene la conexión.

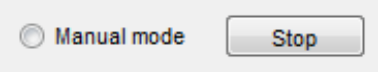

Ilustración 5-VIII Manual mode

Una vez elegida la opción temporal, se debe pulsar sobre Show Live para comenzar la comunicación y adquisición de datos.

Pulsando esta opción se inicia el programa principal para la adquisición de datos, que lo primero que hace es a través de una función externa abrir el puerto serie para proceder a la lectura.

El resto del código no es más que un bucle en el que se incrementa los segundos transcurridos con cada pasada (ya que el GPS envía la información cada segundo) y tras leer los datos del puerto serie, otra función externa, los almacena en las variables correspondientes en cada lectura y muestra sus valores, tanto en los displays digitales como en las gráficas.

Un mismo valor se almacena en dos tipos distintos de variables, esto es debido a que para poder mostrar las variables en tiempo real la variable tiene que ser de tipo global, y a la hora de almacenarlas se realiza en handles.

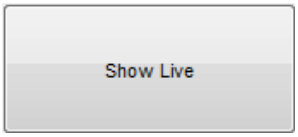

Ilustración 5-IX Show Live

Durante la comunicación para activar o desactivar las gráficas se utilizan una serie de botoneras, que solo las que están activadas se representarán al entrar en el programa principal.

Cuando la casilla esta activada la variable global de la opción respectiva habilita la visualización de su gráfica de valores, al ser preguntada en el programa principal. Igualmente se puede desactivar en cualquier momento de la comunicación.

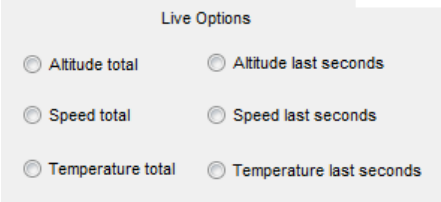

Ilustración 5-X Live Options

En algunas ocasiones se pueden producir determinados errores con la comunicación serie, que impidan cerrar la comunicación, por lo que no es posible volver a abrir otra, y se tenía que recurrir a cerrar el programa totalmente para volver a comenzar. Para solucionar esto se colocó un botón de cierre obligatorio de la conexión serie.

Esto se realiza mediante una instrucción que cierra todos los procesos abiertos.

```
delete(instrfindall)
```

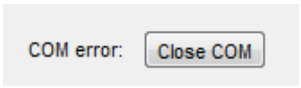

Ilustración 5-XI Close COM

Tras terminar la comunicación, se puede trabajar con esas variables para representarlas gráficamente, para conseguir archivos de texto de salida o para volver a guardar el workspace para un uso posterior en Matlab.

Hay tres tipos de gráficas, una de ellas representada en la función axis que es sobre la interfaz GUI y dos mediante la opción figure, que proporciona un mayor aprovechamiento, ya que incluye opciones como zoom o markers.

Para poder mostrar los valores, primeramente hay que elegir la variable que se quiere visualizar, para ello se elige en el desplegable situado en la parte izquierda la opción que se quiera.

Ilustración 5-XII Variable a visualizar

Esta elección de modo de representación se realiza mediante las distintas opciones:

- Análisis rápido: En esta opción entre las variables a visualizar se muestra una, acompañada de la fecha de comienzo de adquisición de los datos y la hora de inicio y finalización.

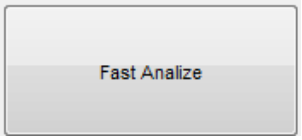

Ilustración 5-XIII Fast Analize

- Análisis detallado: tan solo muestra los valores elegidos mediante el desplegable, pero al ser figure se puede realizar zoom o incorporar markers en la gráfica para su mejor comprensión.

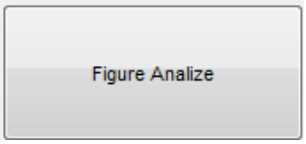

Ilustración 5-XIV Figure Analize

- Visión 3D: mediante una visualización figure, representa los datos seleccionados en unos ejes coordenados mediante latitud y longitud, por lo que se puede observar el perfil del viaje realizado durante la adquisición de los datos.

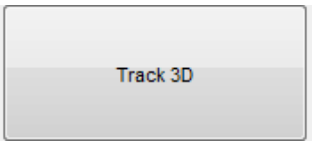

Ilustración 5-XV Track 3D

Para poder almacenar los datos, como archivos de texto hay dos opciones, la primera es crear la trama de datos vista anteriormente, que aúna los valores importantes, y la otra es crear el tipo de trama que utiliza la web www.GPSvisualizer.com para crear sus track, que son las tramas GPS NMEA, pero como el programa ya ha desechado parte de esa información se vuelve a crear dejando valores nulos donde no va información útil.

El procedimiento es tras coger el nombre de la casilla de introducción del texto, se abre el archivo en modo escritura, y con la función fprintf se escribe el texto tanto fijo (cabeceras y espaciados) como variable, que son los datos de las variables.

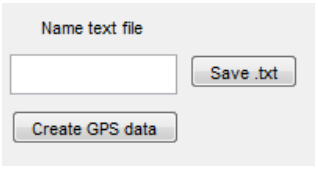

Ilustración 5-XVI Archivos de texto

Para guardar el workspace y que se pueda volver a utilizar por el mismo programa, se debe escribir en la casilla correspondiente el nombre del archivo y dar a guardar o cargar según proceda.

Esto se realiza mediante las siguientes funciones, según corresponda:

```
save (get(handles.textws,'String'))
load (get(handles.textws,'String'))
```

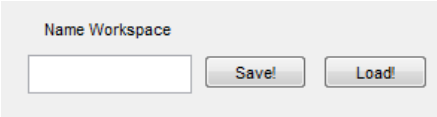

Ilustración 5-XVII Funciones workspace

Por último hay que comentar las posibilidades que ofrece el programa para cambiar los modos de funcionamiento, tienen dos utilidades distintas. Existe la opción de cambiar de programa de funcionamiento en los dispositivos datalink a la vez que se cambia la interfaz del programa al cual se ha cambiado, y la opción de solo cambiar el programa de funcionamiento de los dispositivos, esta opción es para cuando se produce algún problema en la transmisión y la interfaz gráfica y el programa de los datalink no coincide.

Para saber en qué modo de funcionamiento se encuentran los datalink existe la opción de preguntárselo a los dispositivos, mediante State?.

Para poder realizar esto, se abre el puerto serie, se escribe la trama correspondiente que hace a los dispositivos cambiar de modo, ya sea “&u” para User Mode o “&w” para Wait Mode, y se cierra el puerto.

Tras esto se limpian las variables, se cierra la pantalla del modo GPS, y se abre la que se haya elegido al pulsar.

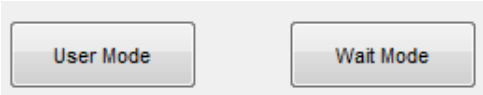

Ilustración 5-XVIII Cambio Completo de modo

Los tres botones de cambio de modo de la parte de arriba, tiene el mismo procedimiento al anterior, a excepción de la limpia de las variables y el cambio de interfaz de usuario.

Para la opción State?, lo que hace es enviar con el procedimiento de abrir puerto serie, enviar “&s”, cerrar puerto y mostrar en la celda blanca el estado en el que se encuentra.

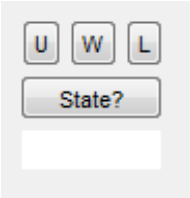

Ilustración 5-XIX Cambio Auxiliar de modo y Solicitud Estado

Tras tener la programación realizada y ver que se recibe todo correctamente, se procede a realizar una prueba con ambos transceptores montados en un coche, por lo que no habrá ningún problema por desapuntamiento de las antenas, simplemente se probará la precisión del GPS.

La imagen que se muestra a continuación corresponde a la que lleva información de posicionamiento en latitud, longitud y altura.

Ilustración 5-XX Track coche (1)

El error que comprende en latitud y longitud es pequeño pero existe, eso se ve en que en la recta que se observa en la parte baja del mapa la traza está muy pegada al carril izquierdo, cuando la conducción se produjo en el otro carril.

Los errores de altitud son mucho más visibles, se observa claramente que la traza desaparece en dos tramos del track, eso es debido a que la altitud que nos da el receptor GPS es menor que la altitud que da el mapa en ese punto.

Para intentar buscar un razonamiento a esos errores de altitud que se producen durante la toma de datos, se realiza otro día distinto otro track exactamente igual y se comparan sus resultados con el primero, obteniendo el siguiente track:

Ilustración 5-XXI Track coche (2)

Se observa en la comparación que hay determinadas zonas que en ambos track se pierde la traza, situándose por debajo del mapa, pero hay otra en la que la traza solo desaparece a la segunda.

Los errores en la altitud al ser coincidente en dos de los tres tramos en los que se pierde, se puede llegar a la conclusión que en esas zonas existe una peor cobertura GPS, y la localización no consigue llegar a ser tan precisa, y se observa más claramente con la altitud que con los datos de latitud y longitud. También influye mucho la precisión con la que el GPS puede llegar a situarse, visto en el ejemplo que daba el fabricante.

5.1.3. Prueba de Modo Localización (Location o GPS Mode):

Tras la finalización de todo el software, incluyendo a todos los modos, y la prueba del correcto funcionamiento del programa, tanto en el lugar de la programación, como en un coche, llevando el emisor de tramas GPS a bordo del coche, y el receptor en un lugar estático conectado al ordenador que mediante Matlab guardará los datos tomados, se procede a instalar el segmento aéreo en un vehículo que pueda permitirnos realizar las pruebas oportunas.

5.1.3.1. Vuelo en parapente (1)

Para que se pueda volar en parapente, deben cubrirse los utensilios de transmisión con plástico acolchado para amortiguar posibles golpes y acoplarlos en una riñonera preparada para esos menesteres, mediante la ayuda de un compañero de trabajo, Daniel Santos Kötting, que vuela en parapente desde hace años, se procedió a la búsqueda de día y sitio para hacer las pruebas.

El sitio que se decidió fue el puerto de Somosierra, al norte de Madrid.

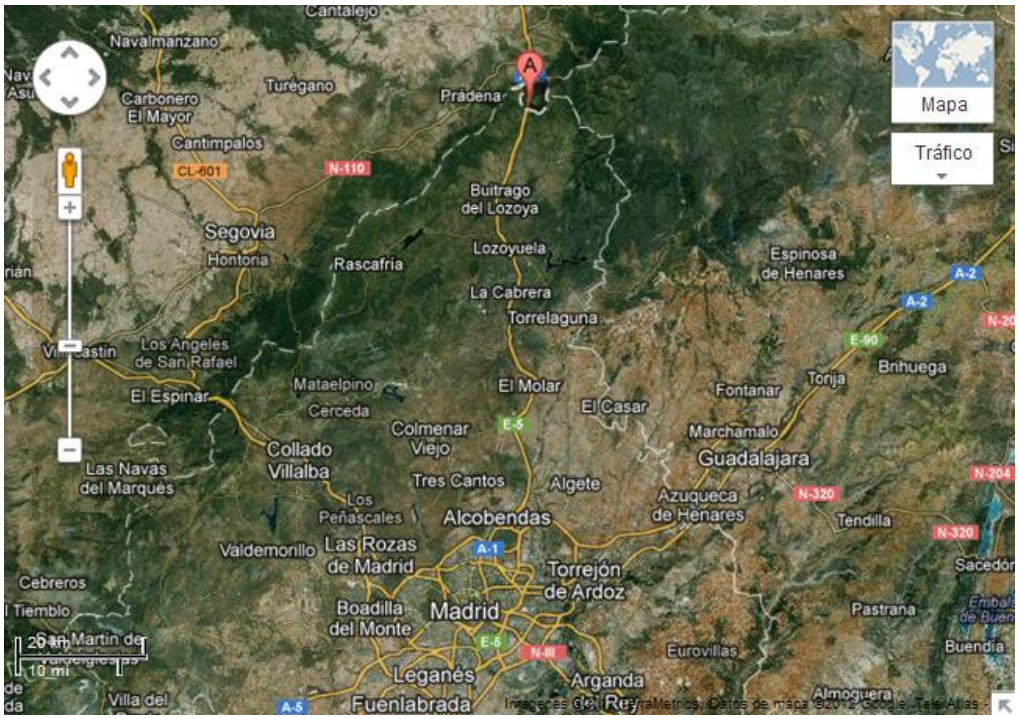

Ilustración 5-XXII Mapa ubicación Somosierra

Un lugar con una buena orografía, y muy visual para el vuelo, llegando a verse Segovia si el vuelo se dirige hacia ese lugar.

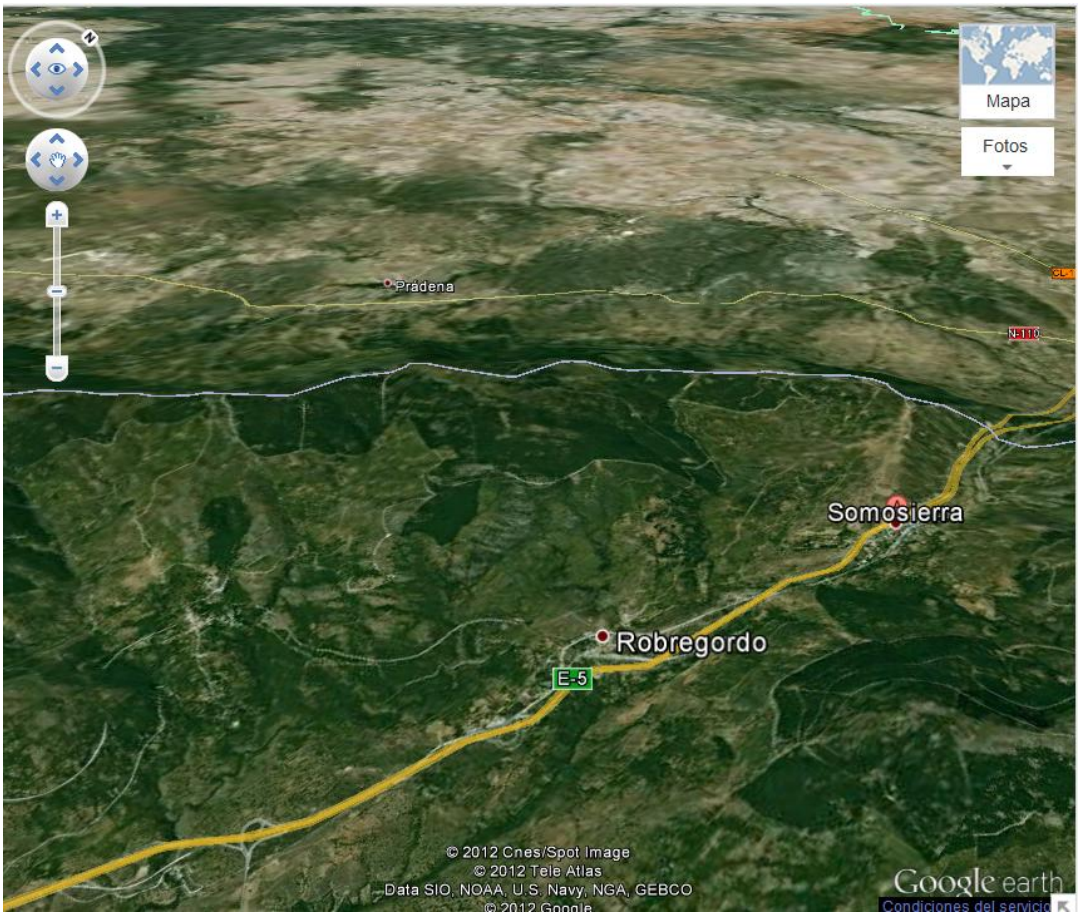

Ilustración 5-XXIII Mapa relieve Somosierra

Con todos los equipos preparados: dos parapentes biplazas, sillas de montar, cascos de protección, ropa de abrigo (está permitido el vuelo hasta algo más de 3.000 metros de altura), buen calzado para el aterrizaje, y el equipo datalink+GPS, protegido y guardado en la mochila de los utensilios, se llego al pueblo de Somosierra.

Debido a las condiciones climatológicas se hizo imposible el vuelo, ya que había grandes rachas de aire, por lo que lo hace muy peligroso, y se tuvo que decidir volver.

Se muestra en la siguiente imagen.

Ilustración 5-XXIV Imagen del lugar de vuelo

5.1.3.2. Vuelo en helicóptero

Por motivos de tiempo, debido a la llegada del verano, y a que las personas que podrían ayudar a realizar las pruebas iban a desplazarse fuera de la Comunidad de Madrid, se decide al día siguiente realizar las pruebas en otro sistema aéreo, menos espectacular a las pruebas, debido a que el vuelo en parapente iba a ir acompañado de un vídeo a bordo, este es un helicóptero pilotado por radiocontrol, en concreto uno de pequeño tamaño (1.5 metros de palas).

Quitando las protecciones contra golpes que envolvían el datalink y el GPS, se observa que se ha producido una rotura en el datalink, en concreto en la base de la antena. En la siguiente foto se observa la pieza de conexión de la antena con la placa de pruebas, que se rompió tanto en el vivo como en las cuatro patas que conectan a masa.

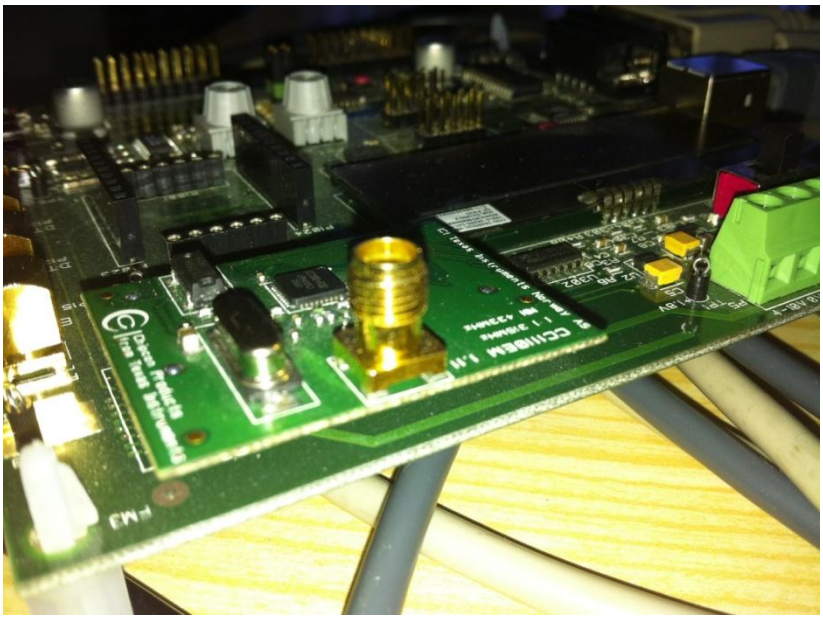

Ilustración 5-XXV Conector antena por defecto

Por el pequeño tamaño disponible para realizar la soldadura, no se puede volver a dejar con la estructura de la original.

Por lo que mediante unos hilos de cobre, se prolonga tanto el vivo como la masa a uno de los lados de la placa, y se fija con silicona.

Pudiendo así resolver el problema, y poder continuar con las pruebas.

Quedando de la siguiente manera:

Ilustración 5-XXVI Conector antena arreglado

El siguiente paso es montar la parte aérea del enlace de datos en el helicóptero, asegurándolos bien, ya que no van resguardados como en la anterior ocasión.

Antes de mostrar la localización del instrumental, se debe comentar que la radio para manejar el helicóptero esta a una frecuencia de 2.4GHz, por lo que no se producen interferencias entre el datalink y el RC.

Se introduce una diferencia frente al montaje del parapente, la antena no irá montada directamente sobre el soporte, para evitar una nueva rotura debido a las vibraciones, ira conectada desde la placa mediante un cable coaxial SMA macho – SMA macho y un adaptador SMA hembra – SMA hembra que acabará conectado con la antena. Este cambio también ayuda a la colocación de los equipos en el helicóptero.

Antes de montar se debe mirar si el helicóptero puede elevar el instrumental debido al peso de este, por lo que se procede al pesaje de los equipos ajenos al helicóptero y el propio helicóptero.

El helicóptero puede aproximadamente con un tercio de su peso, por lo que no existe ningún problema en este aspecto.

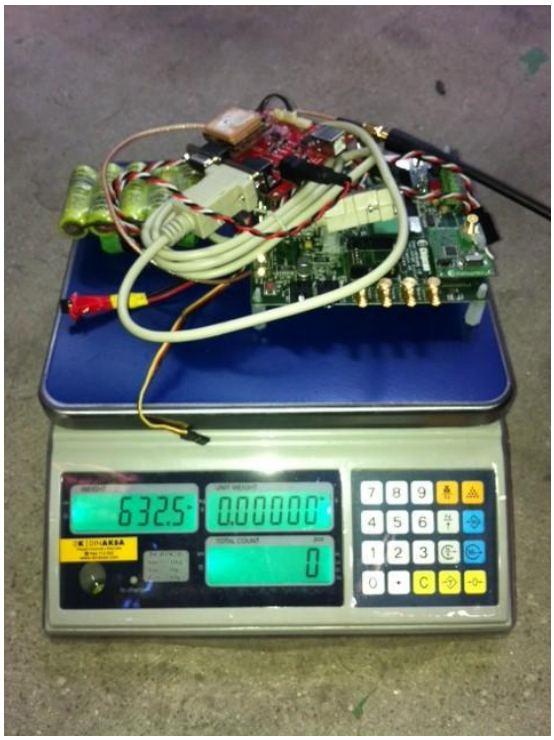

Ilustración 5-XXVII Peso equipo

Ilustración 5-XXVIII Peso helicóptero

En una vista general, la distribución de los equipos es la siguiente:

Ilustración 5-XXIX Montaje en helicóptero (1)

Todas las baterías que se ven en la parte delantera del helicóptero y la de mayor tamaño del centro, son para el uso propio del vehículo, para alimentar la radio y los distintos servos.

También se puede observar otro conjunto de baterías en la parte que se encuentra debajo del rotor, estás son para alimentar todo lo relacionado con el enlace de datos, es un conjunto de dos baterías LiPo (Lithium Polymer), una de 12V que alimenta el datalink, y otra de 9V para la placa del localizador GPS.

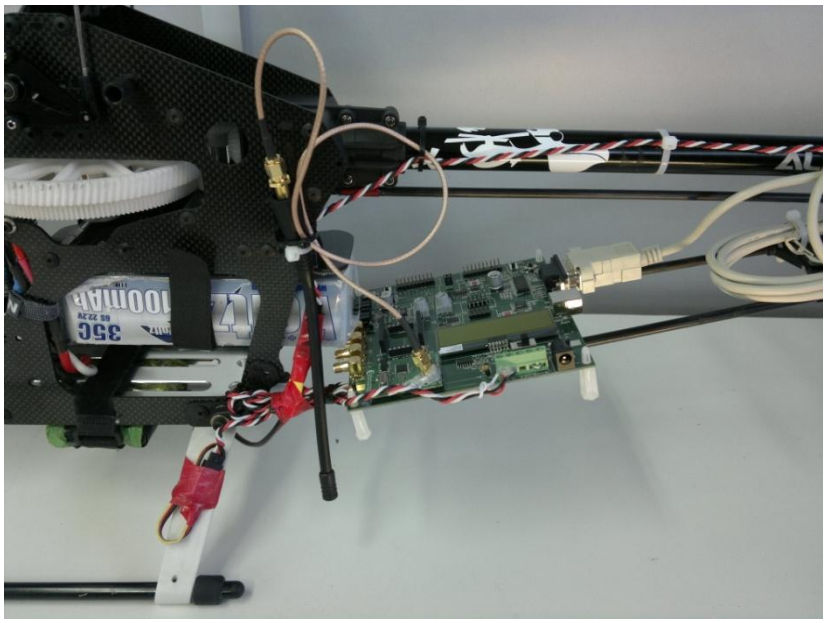

Ilustración 5-XXX Montaje en helicóptero (2)

En la parte trasera, lo más cerca posible al centro de equilibrio del helicóptero, para evitar problemas a la hora de volar, debido a que es el elemento externo con más peso que se va a adaptar, se coloca el datalink, con el cable de prolongación y la antena colocada hacia abajo para intentar evitar pérdidas durante la mayor parte del vuelo.

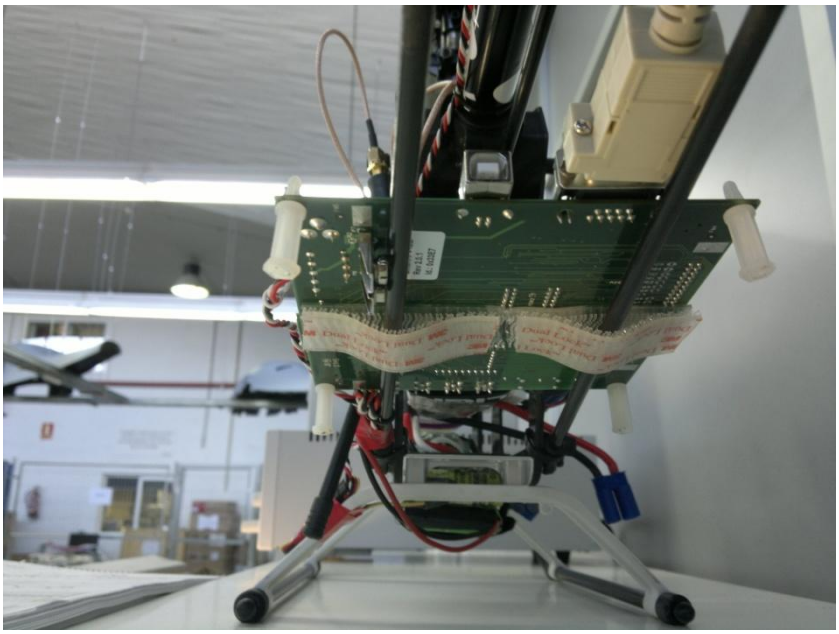

Ilustración 5-XXXI Montaje en helicóptero (3)

Por la parte de abajo se sujeta a la cola mediante cinta adhesiva de doble cara con velcro en la parte interior, quedando totalmente adherido y sin problemas de movimiento ninguno.

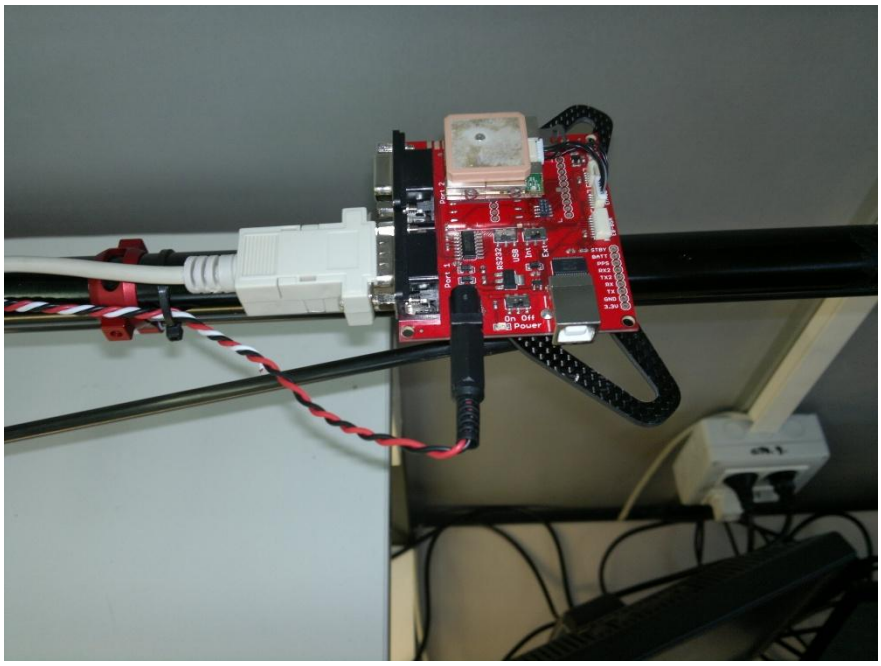

Ilustración 5-XXXII Montaje en helicóptero (4)

La placa de pruebas GPS mas el receptor también están sujetas con la cinta de doble cara, viéndose claramente en esta foto como el cable de alimentación va guiado mediante bridas, al igual que los cables RS232 de comunicaciones.

En la siguiente foto se aprecia el guiado de los cables a lo largo del helicóptero.

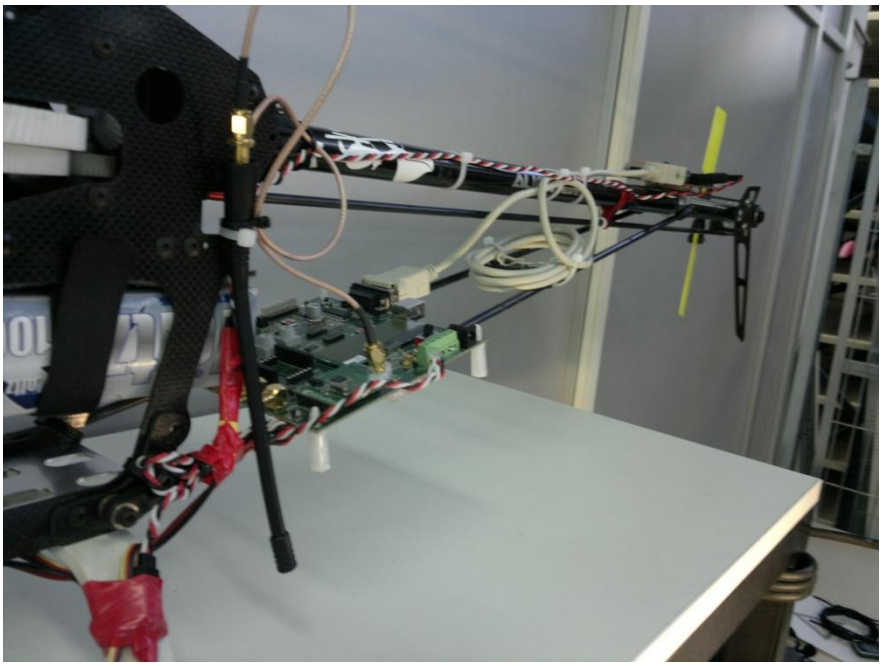

Ilustración 5-XXXIII Montaje en helicóptero (5)

Todos los cables deben de estar bien guiados, sin posibilidad de movimiento, en este aspecto se ha de ser muy cuidadoso, porque podría engancharse con las hélices durante el vuelo.

Una vez realizado el montaje de todo el instrumental, se procede a realizar el vuelo de prueba.

Para ello se cuenta con la colaboración de dos compañeros de Indra Sistemas, Ignacio Díaz de Liaño Gómez, piloto de back up del helicóptero PELICANO y varias veces campeón de España en distintas modalidades de aeromodelismo, y Sergio González Esteban, ingeniero de Telecomunicación, que ayudará al despliegue de los equipos.

El lugar elegido para realizar este vuelo de adquisición de datos, es el polígono industrial Las Monjas, en Torrejón de Ardoz, por su cercanía y que cuenta con varias calles construidas en las que aún no alberga edificios, por lo que no hay prácticamente tráfico.

Ilustración 5-XXXIV Imagen aérea polígono Torrejón

Por lo que estando en el lugar del vuelo se procede a desplegar el equipo.

El helicóptero ya estaba prácticamente operativo, tan solo se tuvieron que colocar las palas en su sitio, y volver a comprobar el estado de las baterías.

Ilustración 5-XXXV Helicóptero preparado para vuelo

Tras colocar el equipo en tierra se comprobó que hubiera comunicación entre los dispositivos, estando colocados a pocos metros, y tras ver que todo funcionaba correctamente, se procedió a un vuelo con la toma de datos.

Ilustración 5-XXXVI Estación terrena para vuelo

Ilustración 5-XXXVII Imagen durante el vuelo

En las siguientes imágenes se ven los resultados obtenidos.

La primera de ellas es una vista desde el suelo alejada, en la que se observa la trayectoria que llevó el helicóptero, con una traza bastante continua, aunque en los giros bruscos que se realizaron si se observan trayectorias rectas, esto puede ser debido a que en esos giros la antena del helicóptero se desapuntaba con la antena en tierra y se perdía la comunicación.

Ilustración 5-XXXVIII Track visto desde el suelo

En las gráficas de los datos adquiridos por Matlab, se observan claramente las caídas del enlace durante las pruebas.

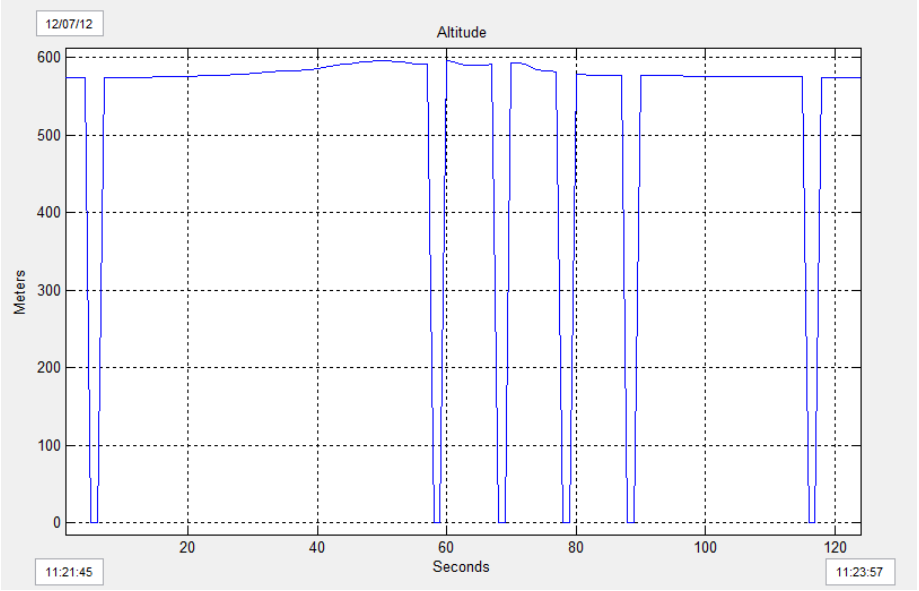

Ilustración 5-XXXIX Datos altitud

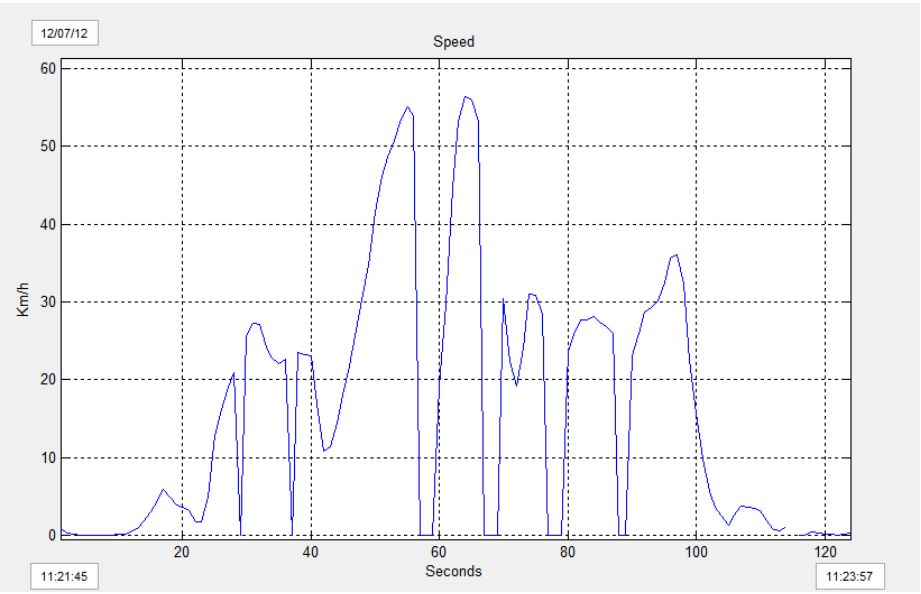

Ilustración 5-XL Datos velocidad

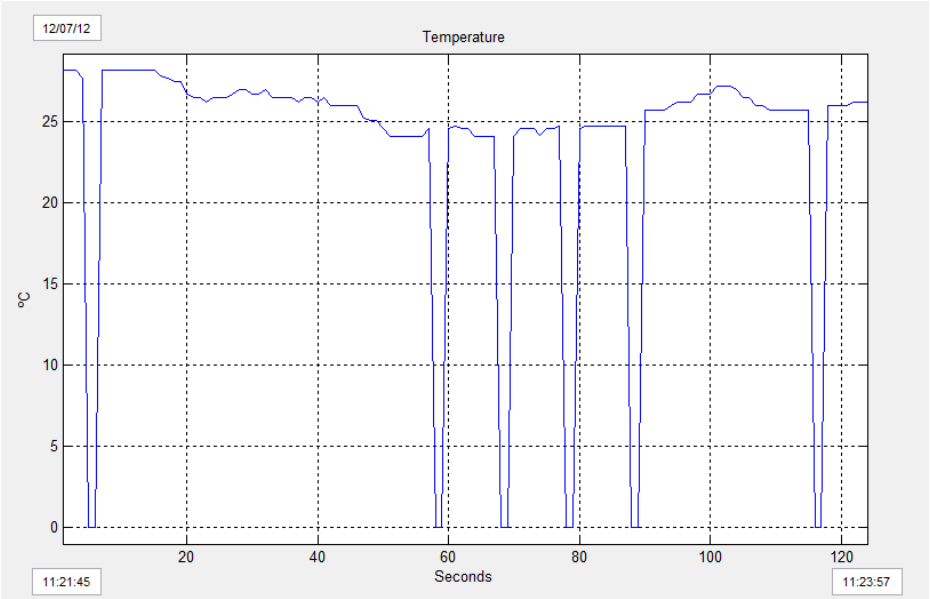

Ilustración 5-XLI Datos temperatura

Igualmente que en el modo de las gráficas con detalle, por ejemplo se observa el caso de la altitud.

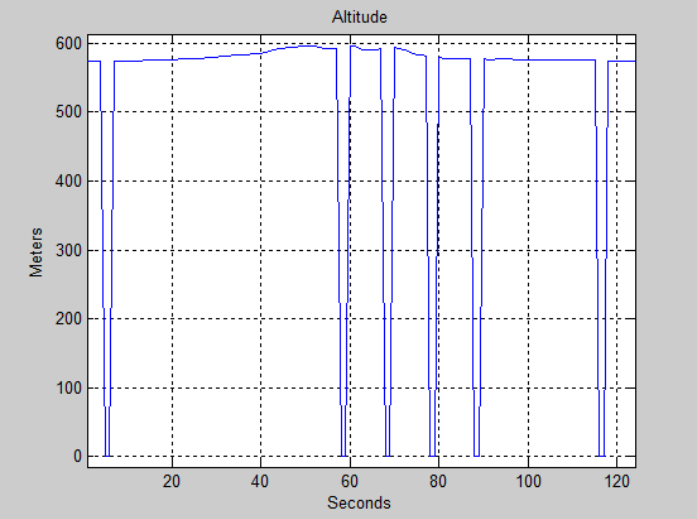

Ilustración 5-XLII Datos altitud (2)

En las siguientes imágenes se ve la comparativa entre los datos con el recorrido total que se realizó (altura incluida), con el que capta los datos sin tener en cuenta la altitud, colocando el track en la superficie del terreno.

Se observa un gran error, debido a que gran parte del vuelo se realizó con un vuelo muy bajo, cercano al suelo, y el error lo sitúo por debajo del mismo.

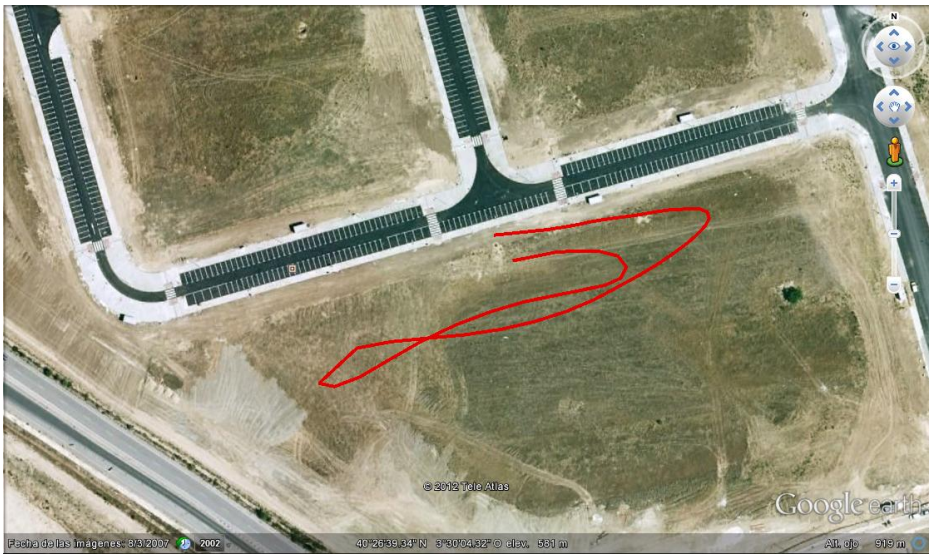

Ilustración 5-XLIII Track con altura

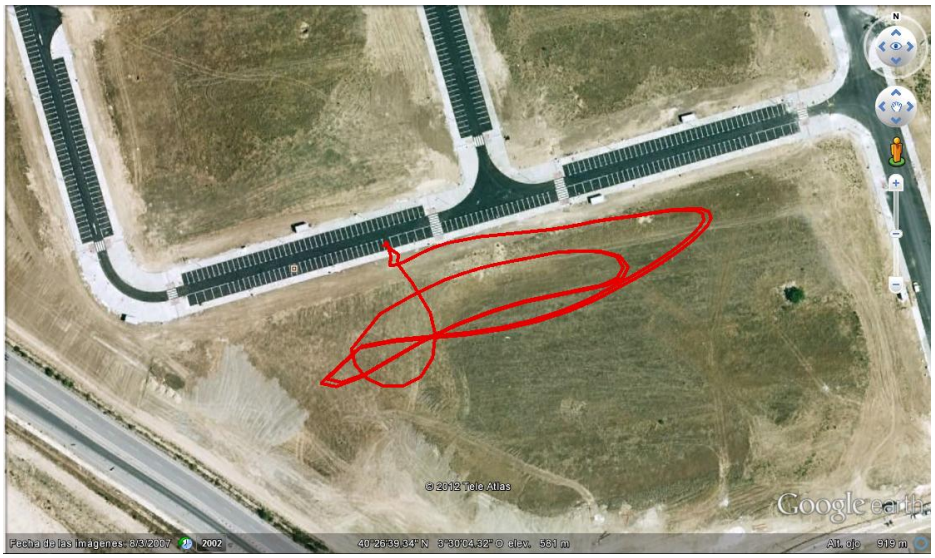

Ilustración 5-XLIV Track sin altura

Cerca de la mitad del tiempo lo colocó por debajo del terreno. Es un error bastante abultado, pero en aplicaciones donde la altura del vuelo está alejada del suelo, el error en altura no es tan determinante.

También se puede observar la trayectoria del vuelo mediante la aplicación Matlab, teniendo los datos sobre las coordenadas GPS y no sobre un mapa.

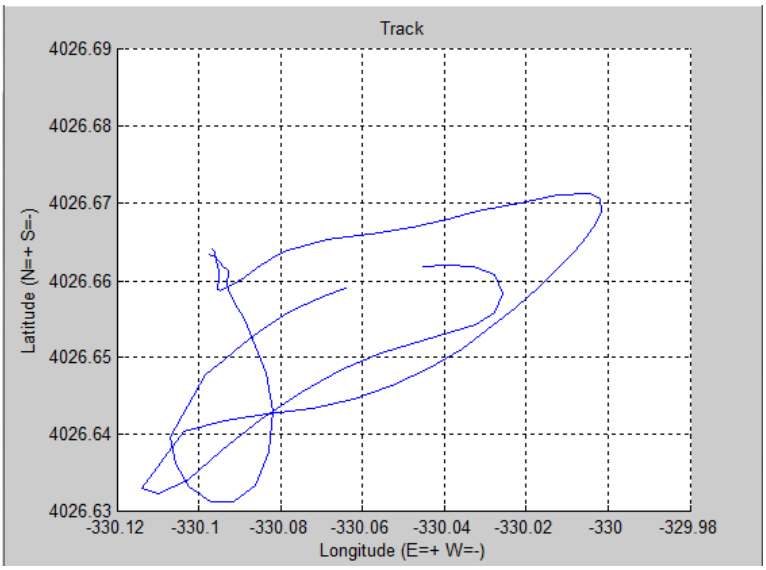

Ilustración 5-XLV Track Matlab

En esta gráfica se observa una discontinuidad, debido a la pérdida de información, que en el modo en el que se representa sobre el mapa, el propio programa une los dos puntos consecutivos.

En esa misma gráfica, rotando la imagen, se observa la altitud del track.

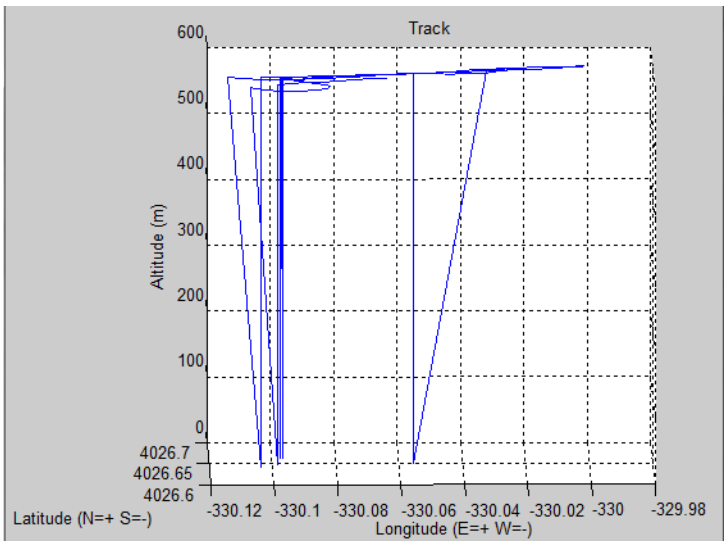

Ilustración 5-XLVI Track vista altura

Al igual que en el caso de la temperatura.

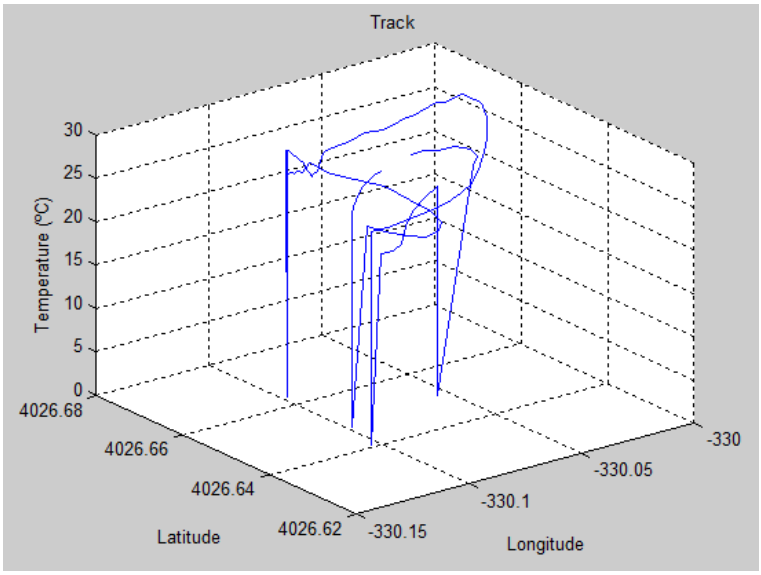

Ilustración 5-XLVII Track con temperatura

5.1.3.3. *Vuelo en parapente (2)*

Tras tiempo intentando buscar de nuevo una fecha para realizar un vuelo en parapente, aunque ya no haría falta para conseguir la adquisición de datos que ya ha sido realizada, se decide hacer para observar posibles problemas que puedan aparecer, capturar nuevos datos, conseguir un video de un vuelo del dispositivo y para probar la sensación que se siente volando, ya que el parapente a utilizar es biplaza.

En esta ocasión el lugar elegido es Alarilla, en Guadalajara, a unos 80 kilómetros de trayecto desde Madrid.

Ilustración 5-XLVIII Mapa ubicación Alarilla

A la hora de llegada al lugar la previsión era de bajo viento, pero no era así, había un viento cercano a 30 Km/h con rachas superiores a 40 Km/h, lo que hacía peligroso intentar volar en esos momentos. Lo que era positivo es que el viento fuera de suroeste, lo que hacía posible el vuelo en la parte su parte sur, que también es la más amplia.

El viento que se crea en las épocas calurosas del año suele ser muy fuerte en las horas centrales del día, debido que el terreno devuelve el calor que acumula, y amaina según se acerca la noche dejando ascendencias sUAVes, éste es un vuelo térmico llamado restitución, por lo que esperando unas horas se podría volar.

Cuando la velocidad del viento fue apta para el vuelo se empezó a desplegar el instrumental, el ala que es de unos 12 metros se dejo plegada en el suelo y se comprobaron todos los instrumentos de seguridad. El equipo datalink+GPS se acopló en la riñonera de los utensilios y el equipo en tierra, ordenador y datalink se coloco en un lugar cercano a la ladera.

Ilustración 5-XLIX Despliegue ala

Ilustración 5-L Equipo terreno

Ilustración 5-LI Equipo aéreo

Cuando todo parecía correcto se procedió a comenzar el vuelo, pero unas rachas de aire frustraban los intentos, hasta que se consiguió.

Ilustración 5-LII Salida vuelo

Con la estación base situada cercana a la ladera, y el vuelo sobrevolando por encima del datalink en tierra, empezaron los problemas en la comunicación, debido a que las antenas no

son isótropas y comenzaron las caídas en la comunicación, llegando incluso a pérdida de sincronismo, por lo que se procedió a volver a reenganchar la conexión mediante el Wait Mode. Realizando el apuntamiento manual del dispositivo en tierra hacia el módulo en aire se mejoraba considerablemente la comunicación, por lo que se volvió a conseguir conexión en el Modo GPS.

Igualmente en determinados momentos se perdía la conexión, lo cual se observa en los datos obtenidos durante el vuelo.

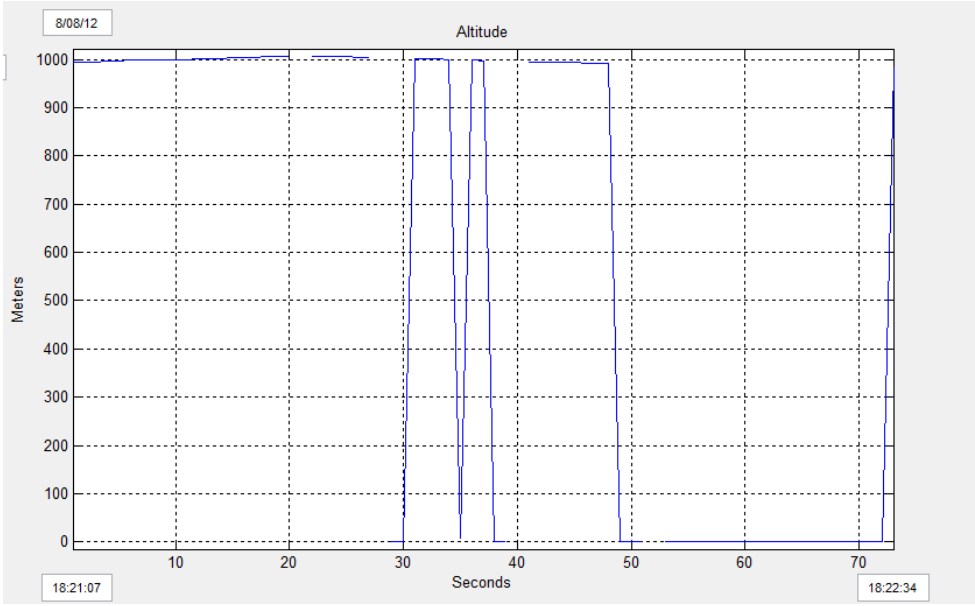

Ilustración 5-LIII Datos altura

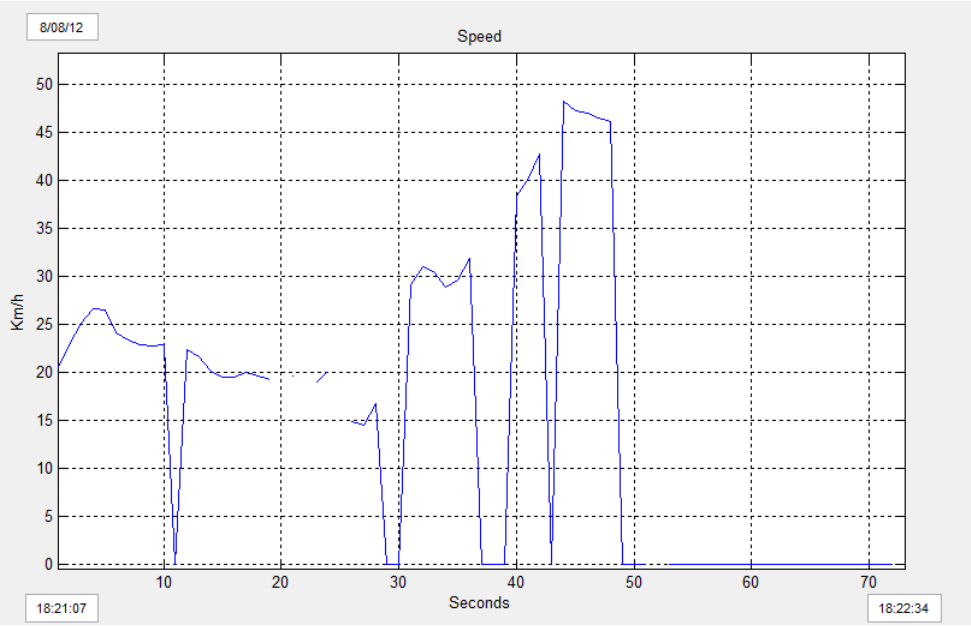

Ilustración 5-LIV Datos velocidad

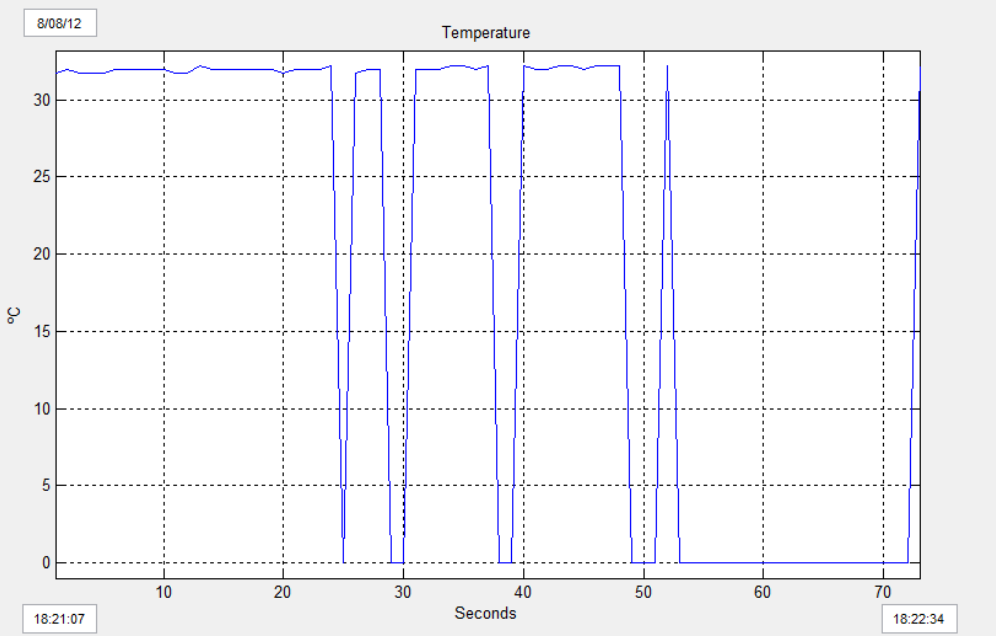

Ilustración 5-LV Datos temperatura

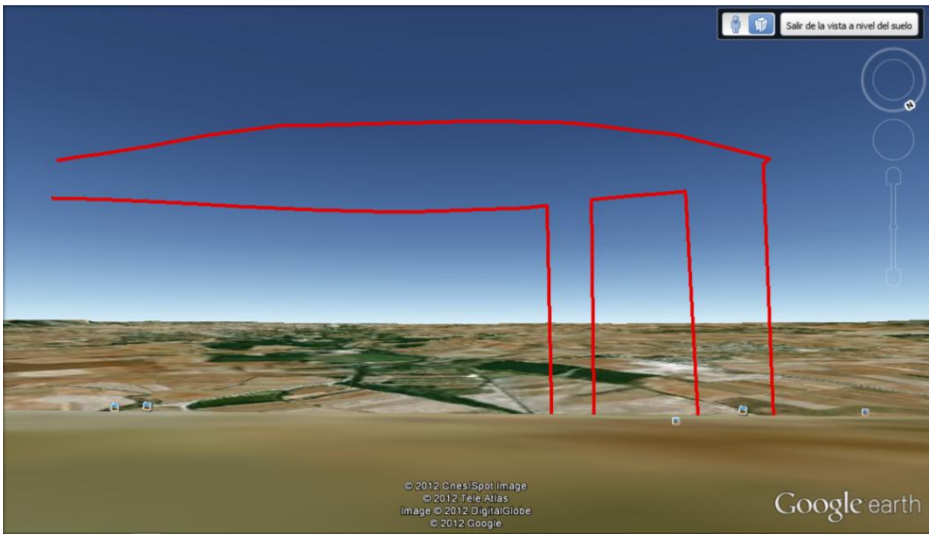

Ilustración 5-LVI Track visto desde el lugar del equipo en tierra

Durante el vuelo se grabo con una cámara incorporada en el casco del piloto del parapente, al igual que se hicieron fotos, como las siguientes.

Ilustración 5-LVII Foto vuelo (1)

Ilustración 5-LVIII Foto vuelo (2)

5.2.Modo Conversación Chat (User Mode)

Este modo es creado por si surgiera la necesidad de interactuar distintos operadores, esto sería de la siguiente manera:

Ilustración 5-LIX Escenario real

Como se observa un datalink se encuentra en el vehículo y dos en bases con operadores, y estos tienen la opción de hablar a tiempo real si fuera necesario, haciendo el datalink del vehículo las veces de relé.

5.2.1. Programación Modo Conversación Chat (User Mode)

La programación que se realiza en los dispositivos cumple con las especificaciones marcadas arriba a excepción de una, tan solo se cuenta con dos dispositivos para hacer las pruebas, por lo que los modos GPS y User no pueden ser utilizados en una sola configuración de los elementos, debido a que están conectados al dispositivo GPS o a un ordenador.

Tendría que haber un tercer dispositivo, que hiciera las veces de repetidor, en cuya programación tuviera ese modo activado, y pasara de un extremo a otro la información.

A continuación se muestra la parte de la programación en la que el dispositivo adquiere el mensaje del ordenador. Cabe explicar que para evitar errores además de tener habilitado el modo User, el mensaje a transmitir debe comenzar con ‘<’ y finalizar con ‘>’.

```
//Trama User
else if (userM == TRUE)
{
	if(writeUart==TRUE)
	{
		AUXleerRS[uartRxIndex] = U0DBUF;
		uartRxIndex++;

//Final de trama
		if(U0DBUF=='>')
		{
			strcpy(USERdata[jU].data,AUXleerRS);
			USERdata[jU].active=1;
			jU++;
			uartRxIndex=0;
			if(jU==3){jU=0;}
			do{
				AUXleerRS[uartRxIndex]='\0';
				uartRxIndex++;
			}while(uartRxIndex<80);
			uartRxIndex=0;
			writeUart=FALSE;
		}
	}
//Comienzo de trama
	else if(U0DBUF=='<')
	{
		uartRxIndex=0;
		AUXleerRS[uartRxIndex] = U0DBUF;
		uartRxIndex++;
		writeUart=TRUE;
	}
}
```

En el programa que se encarga de recibir los datos en el ordenador se deben programar varias funcionalidades y con ciertas características.

Posee al igual que los otros modos una parte que controla lo referido a la comunicación, que son el número de puerto COM, los cambios de modo o el cierre del puerto en caso de error. Cuya programación ya ha sido explicada.

Ilustración 5-LX Opciones COM y modos en User Mode

Las partes características de este modo son las referidas a mantener una conversación con otro operador y poder seguirla en tiempo real a través de un chat, y todo lo referido a ello.

Las principales funciones se activan mediante un botón, y son las siguientes:

- Open Serial: alberga el programa principal que realiza el bucle de funcionamiento, en primer lugar abre el puerto que se le indica, y con los mismos valores de transmisión que en el GPS Mode. El bucle funciona bajo el control de una variable que se activará cuando se pulse sobre Close Serial, y comprueba por este orden las actividades a realizar:
    - Si la tabla que muestra los mensajes está llena, si es así sube todos los mensajes un lugar, excepto el primero que lo borra.
    - Si hay un mensaje preparado para enviar, si es así diferencia entre el mensaje que va a enviar al dispositivo transmisor, al que le añade al principio el carácter '<' y al final '>' para que sepa que es un mensaje de usuario, y el mensaje que va a mostrar en el chat, al que le añade 'S' de send y la hora en la que se ha realizado el envío y lo coloca en el lugar correspondiente de la tabla

de mensajes del chat. Esto último se hace así porque lo guarda en una variable que va almacenando toda la información para posteriormente sacarla en texto.

  - Por último si no hay mensajes para enviar procede a la lectura del puerto serie, en la que aguarda un segundo si no llega a recibir nada. Si recibe el procedimiento es parecido al anterior, pero en este caso añade 'R' y la hora de recepción, colocando el mensaje en el lugar correspondiente del chat y en el último lugar de la variable que servirá para crear el texto.

- Close Serial: es una utilidad con una sola aplicación, activar una variable que cuando se consulta durante el bucle del programa principal termine con la conexión serie que se esté manteniendo en ese momento.
- Send!: tiene un funcionamiento igual que Close Serial, pero interviene otro campo mas, que es la celda correspondiente a la introducción del texto que se quiere enviar, cuando el usuario considere oportuno el texto escrito en esa celda, pulsara sobre Send!, que activará la variable que al ser consultada en el programa principal enviará al otro dispositivo el texto y lo mostrará en la aplicación tanto local como remota.
- Clear Chat: esta función sirve para borrar la tabla del chat que se visualiza en la aplicación, lo que no interviene en la variable que se encarga de almacenar toda la conversación.
- Save.txt: Pulsando sobre esta opción, lo que se hace es crear un archivo de texto con el nombre que se haya escrito en la celda contigua en el que se guarda toda la conversación realizada.
- Estado de la conexión (Connect – No Connect): es un mensaje que aparece sobre los botones de inicio y cierre de la conversación, y muestra el estado de la conexión.

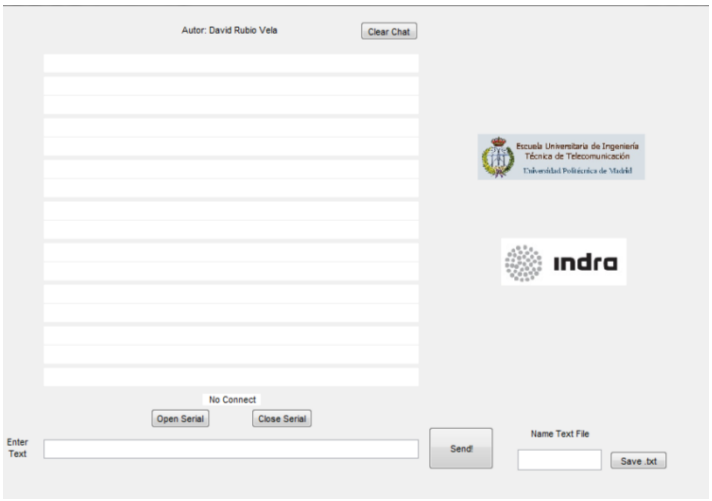

Ilustración 5-LXI Opciones chat User Mode

Una de las características más útiles para entender la conversación que se está realizando, son los caracteres, “R” o “S” que aparece al inicio de cada frase, y la hora de envío o recepción, que aparece al final.

La información de la hora la realiza mediante la función clock, que recoge la hora del ordenador, y ordena los parámetros de la siguiente manera:

Año Mes Día Hora Minutos Segundos

Por lo que la información que se necesita sería la celda 4 y 5 del array.

Para evitar problemas, clock se guarda en otra variable previamente, y se realiza el búcle completo con la creada.

5.2.2. Prueba Modo Conversación Chat (User Mode)

La prueba del programa User Mode se realiza solamente en el lugar donde se ha realizado la programación, debido a que la aplicación a usar el vehículo como relé no se puede realizar, tan solo plantear teóricamente, ya que solo se cuentan con dos dispositivos transceptores.

Abriendo el programa realizado en Matlab, uno de los dos dispositivos se cambia mediante el programa al modo esperado, y en ese momento los dispositivos se sincronizan.

Al preguntar al programa el estado, nos confirma que todo va correcto. Y mirando al display de la placa, se puede comprobar que esta sincronizado.

Ilustración 5-LXII Prueba sincronización

Lo único que se tiene que realizar ahora es una conversación de prueba entre los dos terminales, y por lo que se puede ver que todo funciona correctamente.

Ilustración 5-LXIII Prueba conversación

Tras verificar que todo va correctamente en la transmisión-recepción, se cierra la transmisión, y se saca el comprobante de la conversación en formato texto, quedando lo siguiente:

```
S:hola    Time18:37

R:hola    Time18:38

S:esto es solo una prueba    Time18:38

R:ok    Time18:38

S:aqui va todo corrrecto    Time18:39
```

```
S:alli?   Time18:39

R:tambien   Time18:39

S:muy bien!!   Time18:39
```

5.3.Modo En Espera (Wait Mode)

Esta idea nació para no priorizar entre los dos modos que realmente dan servicio al datalink y comenzar desde un lugar neutro.

La utilidad que tiene este programa es a modo de presentación, donde se observan mejor el autor del proyecto y los organismos colaboradores, y sirve como puente a cualquiera de los otros dos programas. Contando tan solo con las utilidades sencillas para conseguir conectar con los dispositivos para que cambien su modo de trabajo.

5.3.1. Programación Modo En Espera (Wait Mode)

La programación en C es muy básica, durante la operación del modo tan solo da confirmaciones de su estado o cambia a otro modo. Estas confirmaciones se dan tanto en el display del dispositivo en el que muestra los mensajes “wait Mode”, y “sincro rx” en cada cambio de modo si recibe el paquete, como en la aplicación final que al preguntarle por su estado lo muestra en la casilla correspondiente.

En este caso se va a mostrar la parte de programación en C referente al momento en el que recibe la orden de cambiar de modo y activar el de espera. En los otros modos también se sigue un proceso similar.

Tan solo se habilitan dos variables de modo, la de changeM que da prioridad al dispositivo a la hora de realizar un cambio de modo sobre el otro, y el del modo a activar, en este caso waitM.

```
//Activar modo Wait Mode

else if(U0DBUF == 'w')
{
      changeM = TRUE;
      waitM = TRUE;
      userM = FALSE;
      CGPSM = FALSE;
      GPSM = FALSE;
      RGPSM = FALSE;
```

```
        changeC = FALSE;
        uartRxIndex = 0;
        UARTRxFlag = FALSE;
}
```

En Matlab tan solo da estas opciones que son todas referidas a la comunicación con los dispositivos, y ya han sido explicadas anteriormente.

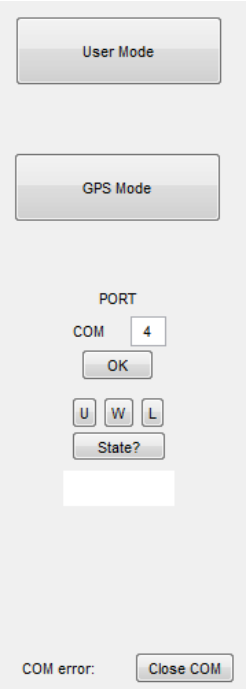

Ilustración 5-LXIVOpciones Wait Mode

6. Manual de Usuario

Para utilizar el programa desarrollado para el datalink es necesario tener un PC con entrada para conector RS232 y tener instalada una versión reciente de Matlab, como es Matlab 2011, lo recomendado es tener de sistema operativo Windows 7, para evitar posibles problemas con la conexión serie, debido a que puede que la programación realizada en dicha versión de Matlab quizás no sea compatible con otras versiones anteriores de Windows.

Se recomienda que se inicie con la ventana de programa perteneciente al Wait Mode, un modo en espera, en el que se inician los dispositivos con el encendido.

6.1.Programa inicial

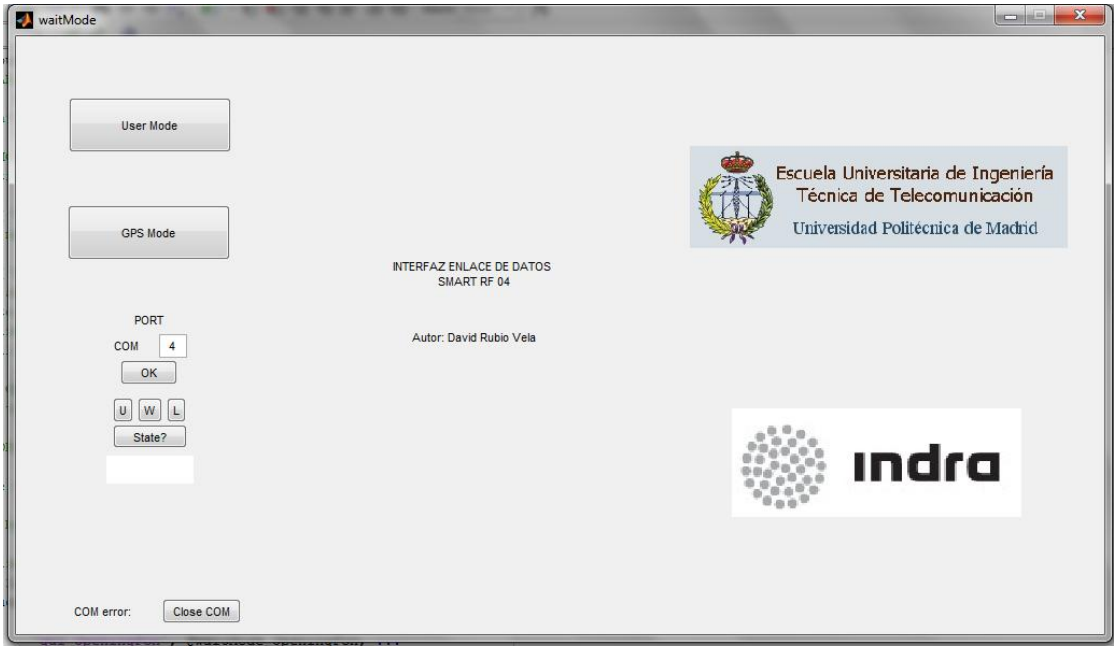

Ilustración 6-I Pantalla Wait Mode

Para cambiar a cualquiera de los dos modos de operación disponibles, solo se tiene que seleccionar la opción deseada, estas son GPS Mode y User Mode.

Ilustración 6-II Botón GPS Mode

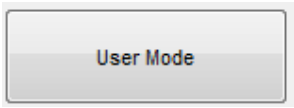

Ilustración 6-III Botón User Mode

Antes de comenzar a explicar los programas específicos de funcionamiento, se comentarán una serie de botones comunes a todos los programas, son los siguientes:

- Opción State?: en cualquier momento se puede consultar el modo en el que están los dispositivos, por si se ha producido algún problema.

Ilustración 6-IV Botón State

- Opción PORT COM: dice al programa en que puerto serie debe leer para recibir los datos. Por defecto se utiliza el puerto 4. Se debe confirmar pulsando sobre OK.

Ilustración 6-V Función Puerto COM

- Opción Close COM: se utiliza para que en caso de problemas con la comunicación se pueda cerrar el puerto serie, para que en una posterior toma de datos esté disponible.

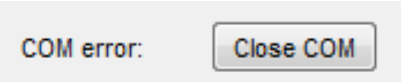

Ilustración 6-VI Función Cerrar COM

- Opciones cambio de modo: Cambia al modo seleccionado, para realizar dicho cambio se recomienda que no se esté adquiriendo datos en dicho momento. Estas opciones son útiles para cuando se produce un error y se está en una ventana de programa, mientras que los dispositivos están trabajando en otro modo.

Ilustración 6-VII Botonera Modos

Dependiendo de la configuración hardware que se tenga, los dos modos de operación diferenciados son los siguientes:

6.2.Terminales finales: PC – GPS.

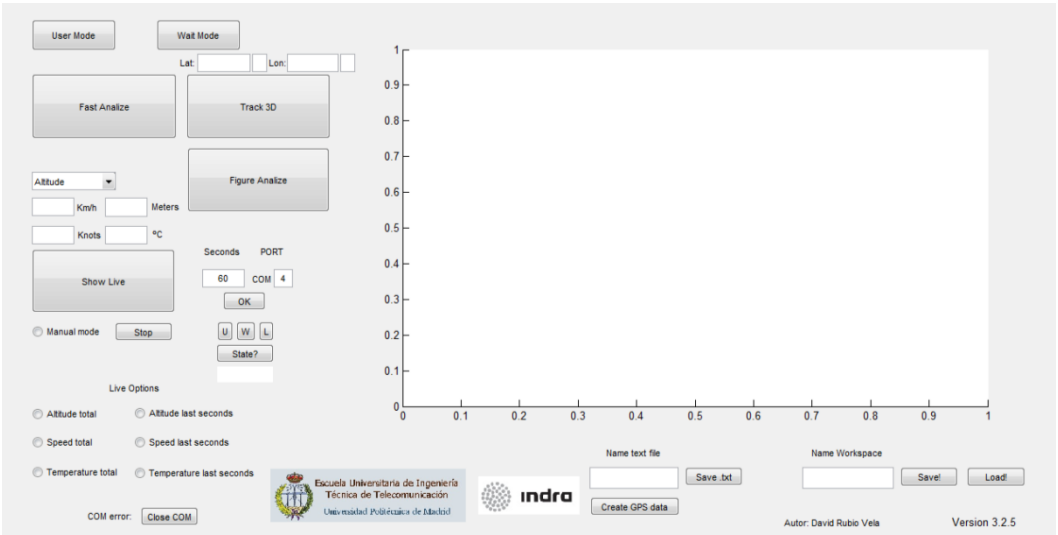

Ilustración 6-VIII Pantalla GPS Mode

Encontrándose en la ventana de GPS Mode y teniendo el modo correctamente en funcionamiento en los dispositivos, para iniciar la recepción de los datos para procesarlos se debe pulsar sobre el botón Show Live.

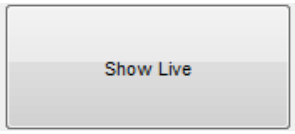

Ilustración 6-IX Botón iniciar toma de datos

6.2.1. Visualización de datos en tiempo real

Para visualizar los datos recibidos en tiempo real, se han de activar sobre la botonera que se encuentra en la parte izquierda de la pantalla, los valores que se desean monitorizar.

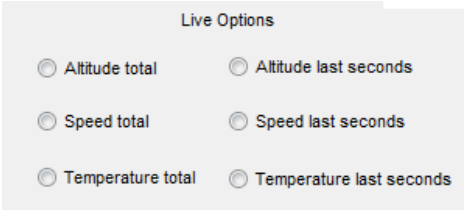

Ilustración 6-X Botonera Tiempo Real

En cualquier momento de la comunicación se pueden activar o desactivar las gráficas visibles, en esa misma botonera.

6.2.2. Visualización posterior de los datos

En el momento que se pulsa la opción Stop o ha terminado el tiempo de adquisición, puede comenzar el estudio de los datos adquirido durante la comunicación en un modo offlive.

Ilustración 6-XI Botón Stop

En un desplegable están las opciones que se pueden visualizar en los distintos tipos de gráficas disponibles.

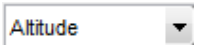

Ilustración 6-XII Opciones Desplegable

Estos se pueden dividir en tres categorías:

- Gráficas estándar: se visualizan los valores de velocidad, altura o temperatura en la misma interfaz del programa, añadiendo la hora de comienzo y finalización de los datos, al igual que la fecha.
  Proporcionan los mismos datos que las de tiempo real, siendo de utilidad para un reconocimiento rápido de los valores.

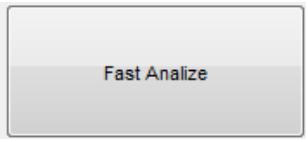

Ilustración 6-XIII Botón gráfica estándar

- Gráficas con detalle: son los mismas variables a visualizar que en el caso anterior pero se representan en una nueva ventana, y la diferencia principal con las anteriores son que dan más detalle en cuanto al zoom aplicable o los markers.

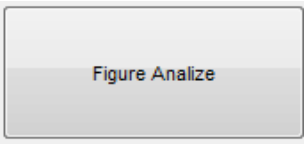

Ilustración 6-XIV Botón gráfica detalle

- Gráfica de trayectoria: es una gráfica en la que se puede visualizar la trayectoria (latitud y longitud) que ha seguido el vehículo durante el track, introduciendo una tercera variable, que son las medidas en tiempo real (velocidad, altitud y temperatura)

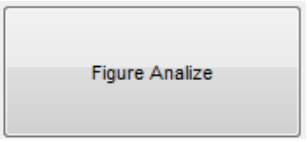

Ilustración 6-XV Botón gráfica trayectoria

6.2.3. Opciones de guardado de los datos

Existe la necesidad de guardar los datos de la comunicación, y esta se puede realizar de varias maneras según la utilidad que se le quiera dar.

- Guardar y cargar workspace: Save, salva los datos de los arrays de las variables, para una posterior carga, Load, en Matlab, de esta manera siempre se podrá realizar la visualización de las gráficas "offlive" de los datos. Recordando que debe de escribir el nombre del workspace, tanto para guardar como para cargar.

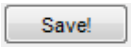

Ilustración 6-XVI Botón guardar

Ilustración 6-XVII Botón cargar

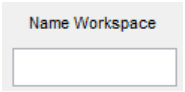

Ilustración 6-XVIII Casilla texto Workspace

- Guardar trama: crea un archivo de texto con tramas propias en la que se incluyen los datos principales de la comunicación.

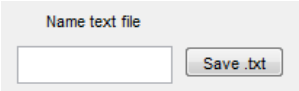

Ilustración 6-XIX Casilla archivo texto

- Crear trama GPS: crea un archivo de texto que servirá para la posterior carga en la página web www.GPSvisualizer.com en la que se crea la imagen en Google Earth o Google Maps entre otras opciones.

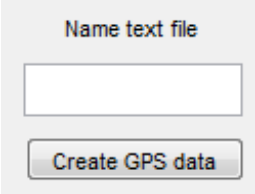

Ilustración 6-XX Casilla archivo GPS

6.2.4. Opciones auxiliares

- Opción Seconds o Manual Mode: Limita el tiempo de la comunicación si se quiere limitar éste, para ello la opción Manual Mode debe estar deshabilitada, si se habilita la comunicación durará hasta que se pare mediante Stop.

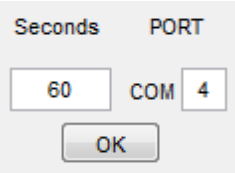

Ilustración 6-XXI Función Puerto COM y Tiempo adquisición

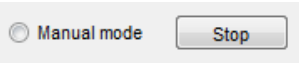

Ilustración 6-XXII Función Manual Mode

- Opción State?, PORT COM, Close COM y cambio de modo, comentadas en el programa inicial.
- Cambiar de modo y programa: se realiza el cambio a los otros dos modos disponibles.

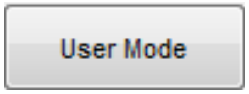

Ilustración 6-XXIII Botón User Mode

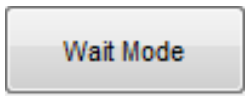

Ilustración 6-XXIV Botón Wait Mode

6.3.Terminales finales: PC –PC

6.3.1. Funcionamiento en tiempo real

Ilustración 6-XXV Pantalla User Mode

- Open Serial: encontrándose en la ventana de User Mode y teniendo el modo correctamente en funcionamiento en los dispositivos, para iniciar la emisión y

recepción de los mensajes se debe pulsar sobre Open Serial, después de cerciorar que el puerto COM es el correcto.

Ilustración 6-XXVI Botón Abrir Conexión Serie

- Enviar mensaje (Send!): a partir de ese momento ya se pueden enviar y recibir mensajes, para enviar se tiene que escribir el texto en la banda blanca colocada en la parte inferior de la ventana, y pulsar sobre Send!.

Ilustración 6-XXVII Casilla texto Chat

- El historial de los últimos mensajes de la conversación, se consulta en la parte central de la pantalla, en el chat donde se registran los últimos 16 mensajes.
- Estado de la comunicación: se encuentra en la parte inferior del chat.

Ilustración 6-XXVIII Conversación Chat

- Clear Chat: para limpiar la pantalla de chat, existe la opción que puede ser utilizada tanto durante una conexión como sin ella.

Ilustración 6-XXIX Botón Clear Chat

- Close Serial: una vez se considere que la comunicación se ha terminado, se debe pulsar sobre este botón.

Ilustración 6-XXX Botón Cerrar Conexión Serie

6.3.2. Opciones auxiliares

- Guardar conversación: existe la opción de guardar la conversación completa en formato texto, esto se realiza escribiendo en la parte inferior, en la banda blanca, el nombre que se desee y pulsar sobre Save.txt.

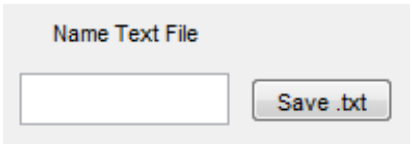

Ilustración 6-XXXI Casilla guardar texto Chat

- Opción State?, PORT COM, Close COM y cambio de modo, comentadas en el programa inicial.
- Cambio de modos: para cambiar a uno de los otros modos, es el mismo procedimiento que en los otros programas. Se debe pulsar sobre el botón correspondiente.

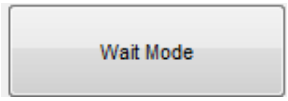

Ilustración 6-XXXII Botón Wait Mode

Ilustración 6-XXXIII Botón GPS Mode

7. Presupuesto

El presupuesto que se va a calcular va a ser el del prototipo utilizado, obviando algunos gastos como el alquiler de un helicóptero RC y un piloto de prueba, como el gasto causado en los desplazamientos para realizar las pruebas, o la compra de dos ordenadores y una licencia Matlab R2011.

En esta primera tabla se muestra el coste de los materiales utilizados, en los cuales no se han incluido los gastos de envío.

Concepto	Precio (dólares)	Precio (euros)
Kit desarrollo Texas	$ 649	515,9 €
Placa desarrollo GPS	$ 39,45	31,4 €
Receptor con antena GPS	$ 59,95	47,6 €
3 Cable RS232	$ 3,3	2,6 €
2 Baterías Lipo	$ 37,7	30 €
Total	$ 789,4	627,5 €

Tabla 1 Presupuesto material prototipo (sin gastos de envío)

En la segunda tabla, se calculan los gastos referidos a personal, los honorarios del ingeniero se calculan en función de un precio/hora de 8€ y un tiempo de trabajo de 300 horas.

El concepto de vuelo en parapente, se ha tomado de un precio estándar de lo que se cobra por un vuelo de unos 20 minutos en un parapente biplaza con video.

Concepto	Precio (dólares)	Precio (euros)
Ingeniero (programador y pruebas)	$ 3.000	2.400 €
Vuelo parapente	$ 100	80 €
Total	$ 3.100	2.480 €

Tabla 2 Presupuesto coste personal

Por lo que el gasto final del proyecto asciende a un total de 3.107,5 €.

Concepto	Precio (dólares)	Precio (euros)
Material	$ 789,4	627,5 €
Mano de obra	$ 3.100	2.480 €
Total	$ 3.889,4	3.107,5 €

Tabla 3 Presupuesto final

8. Conclusiones y líneas de futuro

8.1. Conclusiones GPS Mode

El modo GPS es el que más peso lleva dentro del proyecto, para el que se pensó inicialmente realizar toda la programación, siendo el objetivo buscado.

Tiene varias limitaciones como el del baudrate, en el que quizás se pueden encontrar otras soluciones más aptas para la aplicación final, como el incremento de la tasa en la comunicación en tierra, por lo que se podrían evitar muchas de las pérdidas de transmisión debido a que con la tasa utilizada en ciertas ocasiones, el proceso se veía un poco apurado de tiempo, y existía la posibilidad de que desechara alguna trama por falta de tiempo de proceso. Pero este cambio eliminaría la posibilidad de intercambiar los dispositivos, ya que el código de programación sería diferente.

Otra solución sería la de eliminar la bidireccionalidad de la comunicación, por la que tendría que existir otro dispositivo de comunicaciones paralelo a este, que se denominaría datalink primario, que en un caso real como el que existe en INDRA, tendría que ser de mejores prestaciones, capaz de conseguir mayores anchos de banda, para poder transportar vídeo, de comunicación full-dúplex y con interfaz Ethernet en la conexión con las computadoras, dejando al dispositivo tratado en este proyecto como datalink secundario , y encargado del envío de los parámetros de localización del vehículo aéreo. Este cambio también haría desaparecer el User Mode, que sería trasladado al enlace primario, con las mejoras del modo que nos pueda ofrecer unos dispositivos de mayor calidad.

En cuanto a la prueba realizada en vuelo con el helicóptero fue bastante productiva, ya que no existió ningún problema tanto en el despliegue como en la comunicación, a excepto de las pocas pérdidas que se produjeron bien debido a que en el giro del vehículo se inclinaba y la antena a bordo quedaba desapuntada, o a la falta de tiempo en el tratamiento de los datos.

Con la prueba del parapente, quizás se pudieron mejorar ciertos aspectos, pero por falta de personal de ayuda y de experiencia con el entorno que se encontró allí, ya que nunca se había realizado una comunicación así, hubo quizás menos datos recogidos de los deseados, se podría haber mejorado la comunicación realizando un apuntamiento manual de la antena en tierra, y colocando la antena en vuelo en un lugar fijo, y con mejor visibilidad con su par.

8.2.Conclusiones User Mode

Este modo fue creado como complemento del sistema, para darle más funcionalidad con otra aplicación realista en este ámbito. Sus pruebas se realizaron en el lugar donde se ha programado, por lo que el efecto que puede haber con la distancia no se ha probado, lo que si se ha hecho ha sido intentar saturarlo y no ha habido problema, debido a que el proceso de escritura que tiene programado tampoco da mucha agilidad a la comunicación en el caso de envío de textos cortos, ya que se debe seleccionar la barra de texto, escribir el texto y pulsar sobre el botón de enviar, y ello conlleva un tiempo.

Pero la experiencia de prueba fue muy buena, ya que no se han producido problemas ni en la comunicación, ni en el posterior tratamiento de los datos.

8.3.Líneas futuras dispositivo

Hasta el punto que se ha logrado desarrollar el código de programación con las limitaciones hardware que se tiene, se ha conseguido un enlace viable entre dos puntos distantes con ciertos lazos de seguridad en los que en caso de pérdida de información son capaces de sacar al sistema de esos bucles y darle el poder al usuario para que continúe las comunicaciones con el modo que desee.

Quizás la distancia que se puede conseguir sea insuficiente para la aplicación final que se busca.

Para ello se deberían de introducir ciertas mejoras, como un sistema que mediante un diplexor diera la opción de tener una antena poco direccional y que pueda abarcar muchas direcciones de transmisión para cuando el vehículo aéreo se encontrará en lugares cercanos, haciendo las maniobras de aterrizaje y despegue principalmente, y una antena muy direccional para cuando el vehículo se alejara, para esto también debería de implementarse un sistema de seguimiento de la antena para no perder la comunicación.

Para conseguir ese aumento considerable de cobertura del sistema, también se debería de implementar una cadena de amplificación de la señal tanto para transmitir como para recibir en el entorno de las frecuencias a utilizar.

Otro aspecto sería el de fabricar placas con las utilidades precisas para el funcionamiento, ya que en todo momento se ha trabajado sobre una placa de desarrollo. Ese dispositivo debería ir en una carcasa apta para cumplir los requerimientos de compatibilidades, al igual que los conectores a utilizar, el mallado de los cables y un ordenador en la base, que esté rugerizado, ya que podría sufrir golpes.

Este producto no encaja en el proyecto en el que actualmente está participando el autor del PFC, debido a que es un dispositivo de características muy inferior a los utilizados, pero sí ha

servido como una muy buena toma de contacto con estos dispositivos, y dando tanto trabajo como para poder realizar un proyecto sobre algunas de las utilidades de las que dispone.

9. Bibliografía

Programación C:

J. Carlos López Ardao, “Programación en C”, 2001.

http://es.scribd.com/doc/22283/Manual-De-Programacion-Lenguaje-C

Programación Matlab:

Diego Orlando Barragán Guerrero, “Manual de Interfaz Gráfica de Usuario en Matlab”.

http://es.scribd.com/doc/15532859/MANUAL-DE-GUI-EN-MATLAB

Manuales propios de Matlab:

http://www.mathworks.es/

Información UAVs:

Alejandro Arévalo Alegría, “Vehículos aéreos no tripulados, descripción y capacidades para la obtención de información”.

http://www.escint.cl/php/contenido/UAV%20CORREGIDO.pdf?PHPSESSID=94915b1421537cc2596cb62a334e3d21

Sistemas embebidos:

Juan Manuel Cruz, “Sistemas Embebidos”, Material de clase 2011.

http://laboratorios.fi.uba.ar/lse/

Jesús María Méndez Pérez, “SSEE”, Material de clase 2009.

http://ocw.um.es/ingenierias/sistemas-embebidos/material-de-clase-1/

Manual de usuario CC1110Fx y otros documentos:

http://www.ti.com/tool/cc1110-cc1111dk

Información referente sistema GPS:

http://www.gpsinformation.org/dale/nmea.htm

http://www.sparkfun.com

10. ANEXO

A continuación se muestran los códigos de programación principales en código C y todos los de la aplicación Matlab.

10.1. Código programa main.c datalink

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "hal_main.h"
#include "PFC_main.h"

void main(void)
{
  int owntemp;
  float mytemp;
  int rcvtemp=0;
  int decimtemp;
  float dectemp;
  float unidtemp;
  float avtemp[4];
  float uarttemp=0;
  int itemp=0;
  char tempav[15]="/TEMP,    ,C\r\n";
  char datatemp[3];
  int CounterSinc=0;
  int CounterTX=0;
  int CounterRX=0;
  int PrevCountRX=0;
  int LostPacket=0;
  int CounterLastTX=0;
  int CounterLastRX=0;
  BYTE IDPacketTx=0;
  BYTE IDPacketRx=0;
  int TxTime=0xF0;
  int Timer;
  int i;
  int kU=0;
  int lkU=9; //último valor enviado
  int rU=9; //valor fuera del rango, para inicializar
  int sU=9;
  int kG=0;
  int kk=0;
  int GPSverified = 0;
  int GPScount=0;
  int GPSnodata = 0; //Cuenta cuando no transmite nada
  int seguro=1;
  protocol = TX2;  //modo de transmisor-receptor
  protocol0 = MASTER;
  waitM = TRUE;
  manejarClock(CRYSTAL);

  for(i=0;i<10;i++){
    NMEAdata[i].active=0;
    USERdata[i].active=0;
  }
```

```
INIT_LED1();
INIT_LED3();
INIT_TXSIGNAL();
INIT_RXSIGNAL();
halBuiInitLcd();

TxSettings(StateDataRate);

mode = RADIO_MODE_TX;

// Configure interrupt
INT_PRIORITY();
HAL_INT_ENABLE(INUM_RF, INT_ON);     // Enable RF general interrupt
HAL_INT_ENABLE(INUM_T2, INT_ON);     // Enable Timer2 interrupt
HAL_INT_ENABLE(INUM_URX0, INT_ON);  // Enable USART0 RX general interrupt

uartInitOptions(uartOption1,uartOption2,uartOption3);
uartInitStates(uwaitM,uuserM,uCGPSM,uGPSM,uRGPSM,uotherM);

uartMapPort();
uartInitBitrateGPS();
uartInitProtocol();
uartStartRxForIsr();


CLKCON &= ~0X38;
CLKCON |= 0X28; //101 tick speed
T2PR = 0X09;
T2CTL = 0X12;
RFIM = IRQ_DONE;                     // Mask IRQ_DONE flag only
INT_GLOBAL_ENABLE(INT_ON);           // Enable interrupts globally

do{
  if (mode == RADIO_MODE_TX) {

    if(userM){halBuiLcdUpdate("userM","");}
    else if(GPSM){halBuiLcdUpdate("GPSM","");}
    else if(CGPSM){halBuiLcdUpdate("CGPSM","");}
    else if(waitM){halBuiLcdUpdate("NO MODE","");}
    else if(RGPSM){halBuiLcdUpdate("RGPSM","");}
    else {halBuiLcdUpdate("WAIT","");}

    if(showState){
      if(userM){uart0Send(uuserM,16);}
      else if(GPSM){uart0Send(uGPSM,16);}
      else if(waitM){uart0Send(uwaitM,16);}
      else if(RGPSM){uart0Send(uRGPSM,16);}
      else if(CGPSM){uart0Send(uCGPSM,16);}
      else {uart0Send(uotherM,16);}
      showState=FALSE;
    }


    LED1 = LED_ON;
    TXSIGNAL = SIGNAL_ON;
    if (userM == TRUE) { T2CT = TxTime; GPSverified=0; }
    else if (waitM == TRUE) { T2CT = TxTime; GPSverified=0; }
    else if (CGPSM == TRUE) { T2CT = 0x40; }
    else if(GPSverified == 1) {
      T2CT = 0X05; GPSverified=0; protocol = TX3;}
    else { T2CT = TxTime; }
    CounterLastTX=0;
    GPSnodata=0;

    do{
      dmaRadioSetup(RADIO_MODE_TX);
```

```
mytemp = measureTemp();
owntemp = mytemp; //pasa float a int
sacarFloat(mytemp, &decimtemp, &dectemp, &unidtemp);

if(changeM){CounterSinc = 0;}

if(waitM == TRUE){
  if (CounterSinc == 0){
    radioPktBuffer[0] = PKTLEN;
    radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
    radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
    radioPktBuffer[3] = PACKET_SINC;
    radioPktBuffer[4] = owntemp;
    radioPktBuffer[5] = IDPacketTx;
    radioPktBuffer[6] = T2CT;
    radioPktBuffer[7] = FALSE;
    if(changeM){changeM = FALSE;}
    i=8;
    do{
      radioPktBuffer[i] = 0xEE;
      i++;
    }while(i<=PACKET_LENGTH);
  }
}
else if(userM == TRUE){
  if (CounterSinc == 0){
    radioPktBuffer[0] = PKTLEN;
    radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
    radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
    radioPktBuffer[3] = PACKET_SINC;
    radioPktBuffer[4] = owntemp;
    radioPktBuffer[5] = IDPacketTx;
    radioPktBuffer[6] = T2CT;
    radioPktBuffer[7] = TRUE;
    if(changeM){changeM = FALSE;}
    i=8;
    do{
      radioPktBuffer[i] = 0xEE;
      i++;
    }while(i<=PACKET_LENGTH);
  }
  else if (userM == TRUE){
    radioPktBuffer[0] = PKTLEN;
    radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
    radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
    i=8;

    if (lkU != sU)
    {
      radioPktBuffer[3] = PACKET_DATA;
      radioPktBuffer[6] = lkU; //Ultimo enviado
      radioPktBuffer[7] = rU; //Ultimo recibido
      do{
        radioPktBuffer[i] = USERdata[lkU].data[i-8];
        i++;
      }while((i-8)<=80);

      halBuiLcdUpdate("reenviado",USERdata[lkU].data);

    }
    else
    {
      if (USERdata[kU].active == 1)
      {
        radioPktBuffer[3] = PACKET_DATA;
        radioPktBuffer[6] = kU; //Ultimo enviado
        radioPktBuffer[7] = rU; //Ultimo recibido
```

```
        do{
          radioPktBuffer[i] = USERdata[kU].data[i-8];
          i++;
        }while((i-8)<=80);
        USERdata[kU].active = 0;
        halBuiLcdUpdate("enviado",USERdata[kU].data);
        lkU=kU;
        kU++;
        if(kU==3){kU=0;}
      }
      else
      {
        radioPktBuffer[3] = PACKET_NODATA;
        radioPktBuffer[4] = owntemp;
        radioPktBuffer[5] = IDPacketTx;
        radioPktBuffer[7] = rU;
        do{
          radioPktBuffer[i] = 0xEE;
          i++;
        }while(i<=PACKET_LENGTH);
        halBuiLcdUpdate("confirm","tx");
      }
    }
  }
}
else if(CGPSM == TRUE)
{
  radioPktBuffer[0] = PKTLEN;
  radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
  radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
  radioPktBuffer[3] = PACKET_CGPS;
  radioPktBuffer[6] = T2CT;
  protocol = TX3;
}
else if(GPSM == TRUE)
{
  radioPktBuffer[0] = PKTLEN;
  radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
  radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
  radioPktBuffer[3] = PACKET_GPS;
  radioPktBuffer[4] = owntemp;
  radioPktBuffer[5] = decimtemp;
  radioPktBuffer[6] = FALSE;
  radioPktBuffer[6] = 0xFF; //TIME
  i=8;
  if (NMEAdata[kG].active == 1)
  {
    radioPktBuffer[7] = TRUE;
    do{
      radioPktBuffer[i] = NMEAdata[kG].data[i-8];
      i++;
    }while((i-8)<=80);
    NMEAdata[kG].active = 0;
    if (NMEAdata[kG].data[3] == 'R'){ GPScount++; }
    if(GPScount>=10){
      radioPktBuffer[6] = TRUE;
      GPScount=0; GPSverified = 1;
    }
    else{
      radioPktBuffer[6] = FALSE;
    }
    kG++;
    if(kG==10){kG=0;}
  }
  else{ radioPktBuffer[7] = FALSE; GPSnodata++;}
  halBuiLcdUpdate("enviado",NMEAdata[kG].data);
}
```

```
else if (ChangeDataRate == TRUE){
  if(ChangeCount==0){ T2CT = TxTime/4; }
  radioPktBuffer[0] = PKTLEN;
  radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
  radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
  radioPktBuffer[3] = PACKET_CRATE;
  if(NextDataRate == LOW){radioPktBuffer[4] = LOW;}
  else{radioPktBuffer[4] = HIGH;}
  radioPktBuffer[5] = T2CT;
  radioPktBuffer[6] = ChangeCount;
  i=7;
  do{
    radioPktBuffer[i] = 0xEE;
    i++;
  }while(i<=PACKET_LENGTH);
  ChangeCount++;
}
else if (StateDataRate != NextDataRate){
  if(NextDataRate == LOW){
    if(LowCount == 0){ T2CT = TxTime/4; }
    radioPktBuffer[4] = LOW;
    radioPktBuffer[6] = LowCount;
    LowCount++;
  }
  else if(NextDataRate == HIGH){
    if(HighCount == 0){ T2CT = TxTime/4; }
    radioPktBuffer[4] = HIGH;
    radioPktBuffer[6] = HighCount;
    HighCount++;
  }
  radioPktBuffer[0] = PKTLEN;
  radioPktBuffer[1] = (BYTE) (NETWORK_ID_KEY>>8);     // Network identifier
  radioPktBuffer[2] = (BYTE) NETWORK_ID_KEY;
  radioPktBuffer[3] = PACKET_DRATE;
  radioPktBuffer[5] = T2CT;
  i=7;
  do{
    radioPktBuffer[i] = 0xEE;
    i++;
  }while(i<=PACKET_LENGTH);
}


CounterSinc++;

if (CounterSinc >= 9){ CounterSinc=0; }

// Send the packet
DMAARM |= DMAARM_CHANNEL0;  // Arm DMA channel 0
RFST = STROBE_TX;           // Switch radio to TX
// Wait until the radio transfer is completed,
// and then reset pktSentFlag
while(!interruptFlag);

if (pktSentFlag){
  interruptFlag = FALSE;
  pktSentFlag = FALSE;
  CounterTX++;
  CounterLastTX++;
  IDPacketTx++;
  if(CounterTX>=500){CounterTX=0;}
}

if(LowCount>=5){
  LowCount=0;
  T2CT=0x01;
}
```

```
        else if(HighCount>=5){
          HighCount=0;
          T2CT=0x01;
        }
        else if(ChangeCount>=5){
          ChangeCount = 0;
          StateDataRate = NextDataRate;
          T2CT=0x01;
        }
        else if (GPSverified == 1){

          protocol = TX2;
          T2CT=0x01;
        }
        else if (GPSnodata >= 50){
          GPSnodata=0;
          GPSverified = 1;
          protocol = TX2;
          T2CT=0x01;
        }


      }while(!timer2Flag);
      DMAARM |= DMAARM_ABORT; //Limpio DMA
      RFST = STROBE_IDLE; //Llevo al micro al estado de IDLE
      interruptFlag = FALSE;
      timer2Flag = FALSE;
      LED1 = LED_OFF;
      TXSIGNAL = SIGNAL_OFF;
      radioPktBuffer[3] = PACKET_CLEAR; //Limpio porque el RFST puede acarrear una
interrupcion

      if(protocol == TX1)
      {;
      }
      else if(protocol == TX2)
      {
        if(ChangeDataRate == TRUE){
          ChangeDataRate = FALSE;
          TxSettings(StateDataRate);
        }
        mode = RADIO_MODE_RX;
      }
      else
      {
        mode = RADIO_MODE_RX;
      }
    }

    else if (mode==RADIO_MODE_RX){

      LED3 = LED_ON;
      RXSIGNAL = SIGNAL_ON;
      if(GPSverified == 1) {
        T2CT = 0X10; GPSverified=0; protocol = TX1;}
      else { T2CT = TxTime; }
      PrevCountRX = CounterRX;
      CounterLastRX=0;
      GPSnodata=0;

      if(showState){
        if(userM){uart0Send(uuserM,16);}
        else if(GPSM){uart0Send(uGPSM,16);}
        else if(waitM){uart0Send(uwaitM,16);}
        else if(RGPSM){uart0Send(uRGPSM,16);}
        else if(CGPSM){uart0Send(uCGPSM,16);}
        else {uart0Send(uotherM,16);}
```

```
  showState=FALSE;
}


do{
  // Set up the DMA to move packet data from radio to buffer
  dmaRadioSetup(RADIO_MODE_RX);

  // Start receiving
  DMAARM = DMAARM_CHANNEL0;           // Arm DMA channel 0
  RFST   = STROBE_RX;                 // Switch radio to RX

  while(!interruptFlag);

  if (pktRcvdFlag){
    interruptFlag = FALSE;
    pktRcvdFlag = FALSE;
    CounterRX++;
    CounterLastRX++;

    if(CounterRX>=500){CounterRX=0;} //para probar si se cuelga
    if(LostPacket>=500){LostPacket=0;} //para probar si se cuelga

    if(radioPktBuffer[3] == PACKET_SINC){
      if (pktCheckValidity()){
        if (T2CT < 0){ T2CT = 0x02; };
        T2CT = radioPktBuffer[6];
        halBuiLcdUpdate("sincro","rx");
        if (radioPktBuffer[7]==TRUE){
          if(!changeM){
            if(!userM && !CGPSM){
              userM = TRUE;
              waitM = FALSE;
              GPSM = FALSE;
            }
          }
          else{
            if(userM){changeM = FALSE;}
          }
        }
        else if(radioPktBuffer[7]==FALSE){
          if(!changeM){
            if(!waitM && !CGPSM){
              waitM = TRUE;
              userM = FALSE;
              GPSM = FALSE;
            }
          }
          else{
            if(waitM){changeM = FALSE;}
          }
        }
      }
    }
    else if(radioPktBuffer[3] == PACKET_DATA){
      if (pktCheckValidity()){
        sU = radioPktBuffer[7];
        if(rU != radioPktBuffer[6]){
          rU = radioPktBuffer[6];

          i=8;
          do{
            RECdata.data[i-8]=radioPktBuffer[i];
            i++;
          }while(radioPktBuffer[i-1]!='>');
          uart0escribe(RECdata.data,i);
          i=0;
```

```
            do{
              RECdata.data[i]='\0';
              i++;
            }while(i<80);
          }
        }
      }
      else if(radioPktBuffer[3] == PACKET_NODATA){
        if (pktCheckValidity()){
          sU = radioPktBuffer[7];
        }
      }
      else if(radioPktBuffer[3] == PACKET_CGPS){
        if (pktCheckValidity()){
          GPSM = TRUE;
          CGPSM = FALSE;
          userM = FALSE;
          waitM = FALSE;
          if (T2CT < 0){ T2CT = 0x02; };
          T2CT = radioPktBuffer[6];
          protocol = TX1;
        }
      }
      else if (radioPktBuffer[3] == PACKET_GPS){
        if (pktCheckValidity()){
          if(CGPSM){CGPSM=FALSE; changeM=FALSE; RGPSM=TRUE;} //MODO NEUTRO
          avtemp[itemp] = radioPktBuffer[4] + ((radioPktBuffer[5])*0.1);

          if (T2CT < 0){ T2CT = 0x02; };
          T2CT = 0xFF;
          IDPacketRx++;

          if (radioPktBuffer[7] == TRUE)
          {
            if(RGPSM == TRUE){
              i=8;
              do{
                GPSreceiv.data[i-8]=radioPktBuffer[i];
                i++;
              }while(radioPktBuffer[i-1]!='\n');
              uart0escribe(GPSreceiv.data,80);
              i=40;
              do{
                GPSreceiv.data[i]='\0';
                i++;
              }while(i<80);

              if(GPSreceiv.data[3]=='R'){ //Ultima trama de la secuencia
                uarttemp=avtemp[0];
                averagetemp(uarttemp, datatemp); //Funcion que saca el char del
valor de temperatura
                tempav[6]=datatemp[0];
                tempav[7]=datatemp[1];
                tempav[8]='.';
                tempav[9]=datatemp[2];
                uart0escribe(tempav,15);
              }
            }
            if(radioPktBuffer[6]==TRUE){
              protocol = TX2;
              GPSverified = 1;
              T2CT=0x01;
            }

            itemp++;
            if(itemp==3){itemp=0;}
          }
```

```
            }
          }
          else if(radioPktBuffer[3] == PACKET_DRATE){
            if (pktCheckValidity()){
              if(StateDataRate != radioPktBuffer[4]){
                NextDataRate = radioPktBuffer[4];
                T2CT = radioPktBuffer[5];
                ChangeDataRate = TRUE;
              }
            }
          }
          else if(radioPktBuffer[3] == PACKET_CRATE){
            if (pktCheckValidity()){
              StateDataRate = radioPktBuffer[4];
              T2CT = radioPktBuffer[5];
              ConfirmDataRate = TRUE;
            }
          }
        }
        else
        {
          if(!GPSM){ protocol = TX2; }
        }
      }while(!timer2Flag);
      interruptFlag = FALSE;
      timer2Flag = FALSE;
      LED3 = LED_OFF;
      RXSIGNAL = SIGNAL_OFF;
      DMAARM |= DMAARM_ABORT;
      RFST = STROBE_IDLE;

      if (protocol == TX2){
        if(ConfirmDataRate == TRUE){
          ConfirmDataRate = FALSE;
          TxSettings(StateDataRate);
        }
        if(userWait == TRUE){
          userWait = FALSE;
          userM = TRUE;
        }

        if(!GPSverified)
        {
          Timer=0xFF;
          int tt=0;
          do{
            do{
              Timer--;
            }while(Timer!=0);
            tt++;
          }while(tt<2);
        }
        mode = RADIO_MODE_TX;
      }
      else if (protocol == TX3){
      ;}
      else
      {
        mode = RADIO_MODE_TX;
      }
    }
  }while(1);
}

/*==== INTERRUPT SERVICE ROUTINES ==========================================*/
```

```
/*****************************************************************************
* @fn  rf_IRQ
*
* @brief
*      The only interrupt flag which throws this interrupt is the IRQ_DONE interrupt.
*      So this is the code which runs after a packet has been received or
*      transmitted.
*
* Parameters:
*
* @param  void
*
* @return void
*
*****************************************************************************/
#pragma vector=RF_VECTOR
__interrupt void rf_IRQ(void) {
  RFIF &= ~IRQ_DONE;        // Tx/Rx completed, clear interrupt flag
  S1CON &= ~0x03;           // Clear the general RFIF interrupt registers

  interruptFlag = TRUE;
  if (mode == RADIO_MODE_TX) {
    pktSentFlag = TRUE;
  }
  else if (mode == RADIO_MODE_RX){
    pktRcvdFlag = TRUE;
  }
}

#pragma vector = T2_VECTOR
__interrupt void timer2_interrupt(void) {

  T2IF = FALSE;
  T2CTL &= ~0x40;
  interruptFlag = TRUE;
  timer2Flag = TRUE;
  if (mode == RADIO_MODE_TX){
    ;
  }
  else if (mode == RADIO_MODE_RX){
    ;
  }

}

#pragma vector=URX0_VECTOR
__interrupt void uart0int(void) {

  if(U0DBUF == '&'){
    changeC = TRUE;
  }
  if(changeC)
  {
    //Cambiar Data Rate
    if(U0DBUF == 'c'){
      uartRxIndex=0;
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;
    }
    else if(AUXleerRS[0] == 'c')
    {
      if (U0DBUF == '1')
      {
        uart0Send(uartOption1,4);
        NextDataRate = LOW;
        uartRxIndex = 0;
        UARTRxFlag = FALSE;
```

```
      changeC = FALSE;
    }
    else if (U0DBUF == '2')
    {
      uart0Send(uartOption2,5);
      NextDataRate = HIGH;
      uartRxIndex = 0;
      UARTRxFlag = FALSE;
      changeC = FALSE;
    }
  }
  //Activar modo User Mode
  else if(U0DBUF == 'u')
  {
    uart0Send(uartOption3,16);
    changeM = TRUE;
    userM = TRUE;
    CGPSM = FALSE;
    GPSM = FALSE;
    RGPSM = FALSE;
    waitM = FALSE;
    changeC = FALSE;
    uartRxIndex = 0;
    UARTRxFlag = FALSE;
  }
  //Activar modo Wait Mode
  else if(U0DBUF == 'w')
  {
    changeM = TRUE;
    waitM = TRUE;
    userM = FALSE;
    CGPSM = FALSE;
    GPSM = FALSE;
    RGPSM = FALSE;
    changeC = FALSE;
    uartRxIndex = 0;
    UARTRxFlag = FALSE;
  }
  //Activar modo Location Mode
  else if(U0DBUF == 'l')
  {
    changeM = TRUE;
    CGPSM = TRUE;
    userM = FALSE;
    GPSM = FALSE;
    RGPSM = FALSE;
    waitM = FALSE;
    changeC = FALSE;
    uartRxIndex = 0;
    UARTRxFlag = FALSE;
  }
  //Activar modo Status
  else if (U0DBUF == 's')
  {
    showState = TRUE;
    changeC = FALSE;
  }
  //Modo T2CT
  else if (U0DBUF == 'r')
  {
    T2CT = 0x30;
    changeC = FALSE;
  }
}
//Trama User
else if (userM == TRUE)
{
```

```
    if(writeUart==TRUE)
    {
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;

      if(U0DBUF=='>')
      {
        strcpy(USERdata[jU].data,AUXleerRS);
        USERdata[jU].active=1;

        jU++;
        uartRxIndex=0;
        if(jU==3){jU=0;}

        do{
          AUXleerRS[uartRxIndex]='\0';
          uartRxIndex++;
        }while(uartRxIndex<80);
        uartRxIndex=0;
        writeUart=FALSE;
      }
    }
    else if(U0DBUF=='<')
    {
      uartRxIndex=0;
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;
      writeUart=TRUE;
    }
  }
  //Trama GPS
  else if (GPSM == TRUE)
  {
    if(U0DBUF == '$'){
      uartRxIndex=0;
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;
    }
    else{
      AUXleerRS[uartRxIndex] = U0DBUF;
      uartRxIndex++;
    }

    //Final de trama
    if(U0DBUF == '\n'){
      if (AUXleerRS[4]!='S')
      {
        strcpy(NMEAdata[jG].data,AUXleerRS);
        NMEAdata[jG].active=1;

        jG++;
        uartRxIndex=0;
        if(jG==10){jG=0;}
      }
      do{
        AUXleerRS[uartRxIndex]='\0';
        uartRxIndex++;
      }while(uartRxIndex<80);
      uartRxIndex=0;
    }
  }
}
```

10.2. Código programa txsettings.c datalink

```
#include "TxSettings.h"

void TxSettings(int Option)
{

  //seleccionamos la potencia de salida +10dBm y la frecuencia 433MHz
  PA_TABLE0 = 0xC0;
  FREQ2 = 0x10;               // Actually 433.500 MHz to fit band
  FREQ1 = 0xAC;               // fref=26MHz
  FREQ0 = 0x4E;               // fcarrier=fref·FREQ[23:0]/2exp16

  if (Option==HIGH){
    FSCTRL1 = 0x0C;
    FSCTRL1  = 0x12;   // Frequency synthesizer control.
    FSCTRL0  = 0x00;   // Frequency synthesizer control.
    MDMCFG4  = 0x2D;   // Modem configuration.
    MDMCFG3  = 0x3B;   // Modem configuration.
    MDMCFG2  = 0x13;   // Modem configuration.
    MDMCFG1  = 0x22;   // Modem configuration.
    MDMCFG0  = 0xF8;   // Modem configuration.
    DEVIATN  = 0x62;   // Modem deviation setting (when FSK modulation is enabled).
    FREND1   = 0xB6;   // Front end RX configuration.
    FREND0   = 0x10;   // Front end RX configuration.
    MCSM0    = 0x18;   // Main Radio Control State Machine configuration.
    FOCCFG   = 0x1D;   // Frequency Offset Compensation Configuration.
    BSCFG    = 0x1C;   // Bit synchronization Configuration.
    AGCCTRL2 = 0xC7;   // AGC control.
    AGCCTRL1 = 0x00;   // AGC control.
    AGCCTRL0 = 0xB0;   // AGC control.
    FSCAL3   = 0xEA;   // Frequency synthesizer calibration.
    // Freq. dependent value: 433 MHz band vs 868/915 MHz band:
    FSCAL2 =  0x0A;
    FSCAL0   = 0x1F;   // Frequency synthesizer calibration.
    TEST2    = 0x88;   // Various test settings.
    TEST1    = 0x31;   // Various test settings.
    TEST0    = 0x09;   // Various test settings.
    PKTLEN = PACKET_HIGH;
  }

  else if(Option==LOW){
    // Settings from SmartRFStudio for CC1110, VERSION == 0x03
    // 38.4 kBaud, GFSK modulation, 100 kHz RX filter bandwidth.
    FSCTRL1  = 0x06;   // Frequency synthesizer control.
    FSCTRL0  = 0x00;   // Frequency synthesizer control.
    MDMCFG4  = 0xCA;   // Modem configuration.
    MDMCFG3  = 0x83;   // Modem configuration.
    MDMCFG2  = 0x13;   // Modem configuration.
    MDMCFG1  = 0x22;   // Modem configuration.
    MDMCFG0  = 0xF8;   // Modem configuration.
    DEVIATN  = 0x34;   // Modem deviation setting (when FSK modulation is enabled).
    FREND1   = 0x56;   // Front end RX configuration.
    FREND0   = 0x10;   // Front end RX configuration.
    MCSM0    = 0x18;   // Main Radio Control State Machine configuration.
    FOCCFG   = 0x16;   // Frequency Offset Compensation Configuration.
    BSCFG    = 0x6C;   // Bit synchronization Configuration.
    AGCCTRL2 = 0x43;   // AGC control.
    AGCCTRL1 = 0x40;   // AGC control.
    AGCCTRL0 = 0x91;   // AGC control.
    FSCAL3   = 0xE9;   // Frequency synthesizer calibration.

    FSCAL2   = 0x0A;   // Frecuencia elegida de dos opciones, segun la frec
    FSCAL0   = 0x1F;   // Frequency synthesizer calibration.
    TEST2    = 0x81;   // Various test settings.
```

```
    TEST1    = 0x35;   // Various test settings.
    TEST0    = 0x09;   // Various test settings.
    PKTLEN = PACKET_LOW;
  }

  CHANNR   = 0x00;            // Channel number.
  MCSM1 = 0x30;               // Main Radio Control State Machine configuration. si esta
por debajo de RSSI se refleja a menos que este recibiendo un paquete
  IOCFG2 = 0x0B;              // GDO2 output pin configuration. P1_7
  IOCFG0 = 0x06;              // GDO0 output pin configuration. Sync word. P1_6
  PKTCTRL1 = 0x04;            // Packet automation control. añade al paquete datos RSSI,
LQI y bandera CRC OK
  PKTCTRL0 = 0x45;                // Packet automation control. Data whitening on. CRC
enable. Longitud paquete variable
  ADDR = 0x00;                // Device address. Not used.
}
```

10.3. Código programa showtemp.c datalink

```
#include <stdio.h>
#include <math.h>
#include "showTemp.h"

BYTE protocol;
BYTE protocol0;

void sacarTemp (int Temp, int *Dec, int *Unid, int *Signed)
{
  int cont=0;
  int ok=0;
  if (Temp > 127 || Temp < 0){
      Temp = ~Temp;
      Temp&=0xFF;
      Temp++;
      *Signed=1;
    }
  do{
    if (Temp < cont){
      cont = cont-10;
      *Dec = cont/10;
      *Unid = Temp-cont;
      ok=1;
    }
    cont = cont+10;
  }while (ok==0);
}

void sacarNum(int Num, int *Cent, int *Dec, int *Unid)
{
  int cont=0;
  int ok=0;
  do{
    if (Num < cont){
      cont = cont-10;
      if (cont >= 100){
        *Cent = cont/100;
        cont = Num%100;
        *Dec = cont/10;
        *Unid = cont%10;
        ok = 1;
      }
      else{
        *Cent = 0;
```

```
        *Dec = cont/10;
        *Unid = Num%10;
        ok = 1;
      }
    }
    cont = cont+10;
  }while (ok==0);
}

void sacarFloat(float Num, int *decim, float *Dec, float *Unid)
{
  int cont=0;
  int ok=0;
  int ok2=0;
  do{
    if (Num < cont){
      cont = cont-10;

      *Dec=cont/10;
      Num=Num-cont;
      cont=0;
      do{
        if((Num-cont)<1)
        {
          *Unid=cont;
          ok2=1;
        }
        cont++;
      }while (ok2==0);
      cont--;
      Num=Num-cont;
      *decim=10*Num;
      ok=1;

    }
    cont = cont+10;
  }while (ok==0);
}

void pasarachar(int *num)
{
  switch(*num){
  case 0:*num=0x30;break;
  case 1:*num=0x31;break;
  case 2:*num=0x32;break;
  case 3:*num=0x33;break;
  case 4:*num=0x34;break;
  case 5:*num=0x35;break;
  case 6:*num=0x36;break;
  case 7:*num=0x37;break;
  case 8:*num=0x38;break;
  case 9:*num=0x39;break;
  default:*num=0x3F;break;
  }
}

void SAMPLE_TEMP_SENSOR(unsigned int *v)
{
  //Lee el dato de la entrada ADC del sensor de temperatura
  ADCCON2 = 0x3E;
  ADCCON1 = 0x73;
  while(!(ADCCON1 & 0x80));
  *v = ADCL;
  *v |= (((unsigned int)ADCH) << 8);
}

float measureTemp(void)
```

```
{
  unsigned int adcValue=0;
  float outputVoltage;
  SAMPLE_TEMP_SENSOR(&adcValue);

  // Note that the conversion result always resides in MSB section of ADCH:ADCL
  adcValue >>= 4; // Shift 4 due to 12 bits resolution
  outputVoltage = adcValue * CONST;  //0.61065
  return ((outputVoltage - 760) / TEMP_COEFF);
}

void showTemp(int temp1, int temp2)
{
  char TempChar[16]="My  Temp :    C";
  char TempCharRcv[16]="Its Temp :    C";
  char TempChar1[16]="";
  char TempChar2[16]="";
  int Unid;
  int Dec;
  int Signed=0;
  sacarTemp (temp1, &Dec, &Unid, &Signed);

  pasarachar(&Unid);
  pasarachar(&Dec);

  sprintf(TempChar1,"%c", Dec);
  sprintf(TempChar2,"%c", Unid);

  if (Signed == 1){
    TempChar[10]='-';
  }
  Signed = 0;
  TempChar[11]=TempChar1[0];
  TempChar[12]=TempChar2[0];

  sacarTemp (temp2, &Dec, &Unid, &Signed);

  pasarachar(&Unid);
  pasarachar(&Dec);

  sprintf(TempChar1,"%c", Dec);
  sprintf(TempChar2,"%c", Unid);

  if (Signed == 1){
    TempCharRcv[10]='-';
  }
  Signed=0;
  TempCharRcv[11]=TempChar1[0];
  TempCharRcv[12]=TempChar2[0];

  if (protocol == TX1){
    halBuiLcdUpdate(TempChar, "Mode: OnlyTx");
  }
  else if(protocol == TX2){
    halBuiLcdUpdate(TempChar, TempCharRcv);
  }
  else if(protocol == TX3){
    halBuiLcdUpdate("Mode: OnlyRx", TempCharRcv);
  }

}

void showPacket(int dato1, int temp1, int temp2, int dato2, int dato3, int dato4)
{
  char CTX[16]="Tx    T:  C     ";
  char CRX[16]="Rx    T:  C     ";
  char Digit1[16]="";
```

```
char Digit2[16]="";
char Digit3[16]="";
int Unid;
int Dec;
int Cent;
int Signed=0;

sacarNum(dato1, &Cent, &Dec, &Unid);

pasarachar(&Unid);
pasarachar(&Dec);
pasarachar(&Cent);

sprintf(Digit1,"%c", Unid);
sprintf(Digit2,"%c", Dec);
sprintf(Digit3,"%c", Cent);

CTX[2]=Digit3[0];
CTX[3]=Digit2[0];
CTX[4]=Digit1[0];

sacarTemp (temp1, &Dec, &Unid, &Signed);

pasarachar(&Unid);
pasarachar(&Dec);

sprintf(Digit1,"%c", Unid);
sprintf(Digit2,"%c", Dec);

if (Signed == 1){
  CTX[7]='-';
}
CTX[8]=Digit2[0];
CTX[9]=Digit1[0];
Signed=0;

sacarTemp (temp2, &Dec, &Unid, &Signed);

pasarachar(&Unid);
pasarachar(&Dec);

sprintf(Digit1,"%c", Unid);
sprintf(Digit2,"%c", Dec);

if (Signed == 1){
  CRX[7]='-';
}
CRX[8]=Digit2[0];
CRX[9]=Digit1[0];

sacarNum(dato2, &Cent, &Dec, &Unid);

pasarachar(&Unid);
pasarachar(&Dec);
pasarachar(&Cent);

sprintf(Digit1,"%c", Unid);
sprintf(Digit2,"%c", Dec);
sprintf(Digit3,"%c", Cent);

CRX[2]=Digit3[0];
CRX[3]=Digit2[0];
CRX[4]=Digit1[0];

sacarNum(dato3, &Cent, &Dec, &Unid);

pasarachar(&Unid);
```

```
  pasarachar(&Dec);
  pasarachar(&Cent);

  sprintf(Digit1,"%c", Unid);
  sprintf(Digit2,"%c", Dec);
  sprintf(Digit3,"%c", Cent);

  CTX[13]=Digit3[0];
  CTX[14]=Digit2[0];
  CTX[15]=Digit1[0];

  sacarNum(dato4, &Cent, &Dec, &Unid);

  pasarachar(&Unid);
  pasarachar(&Dec);
  pasarachar(&Cent);

  sprintf(Digit1,"%c", Unid);
  sprintf(Digit2,"%c", Dec);
  sprintf(Digit3,"%c", Cent);

  CRX[13]=Digit3[0];
  CRX[14]=Digit2[0];
  CRX[15]=Digit1[0];

  halBuiLcdUpdate(CTX, CRX);

}

void showUart(WORD* info)
{
  char frase[16] = "Informacion:   ";
  char inf[16] = "";
  int i=0;
  do{ inf[i]=info[i]; i++; }while(i<16);

  halBuiLcdUpdate(frase, inf);
}

void averagetemp(float temp, char value[3])
{
  char Digit1[2]="";
  char Digit2[2]="";
  char Digit3[2]="";
  int Unid;
  int Dec;
  int decim;
  float fUnid;
  float fDec;
  float fdecim;

  sacarFloat(temp, &decim, &fDec, &fUnid);
  Unid=fUnid; Dec=fDec;
  pasarachar(&Unid);
  pasarachar(&Dec);
  pasarachar(&decim);

  sprintf(Digit1,"%c", Unid);
  sprintf(Digit2,"%c", Dec);
  sprintf(Digit3,"%c", decim);

  value[0]=Digit2[0];
  value[1]=Digit1[0];
  value[2]=Digit3[0];
}
```

10.4. Código programa waitMode.m

```
function varargout = waitMode(varargin)
% WAITMODE MATLAB code for waitMode.fig
%      WAITMODE, by itself, creates a new WAITMODE or raises the existing
%      singleton*.
%
%      H = WAITMODE returns the handle to a new WAITMODE or the handle to
%      the existing singleton*.
%
%      WAITMODE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in WAITMODE.M with the given input arguments.
%
%      WAITMODE('Property','Value',...) creates a new WAITMODE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before waitMode_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to waitMode_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help waitMode

% Last Modified by GUIDE v2.5 19-Jun-2012 16:26:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @waitMode_OpeningFcn, ...
                   'gui_OutputFcn',  @waitMode_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before waitMode is made visible.
function waitMode_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to waitMode (see VARARGIN)

% Choose default command line output for waitMode
handles.output = hObject;

%%%% Iicialización globales %%%%

global port
port = 'COM4'; %puerto por defecto



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Inicialización imágenes corporativas %%%%

axes(handles.axes1)
background = imread('euitt','jpg');
axis off;
imshow(background);
```

```
axes(handles.axes2)
background = imread('indra','jpg');
axis off;
imshow(background);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes waitMode wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = waitMode_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in CloseCOM.
function CloseCOM_Callback(hObject, eventdata, handles)
% hObject    handle to CloseCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

delete(instrfindall)




function PortTag_Callback(hObject, eventdata, handles)
% hObject    handle to PortTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of PortTag as text
%        str2double(get(hObject,'String')) returns contents of PortTag as a double


% --- Executes during object creation, after setting all properties.
function PortTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PortTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in OKTag.
function OKTag_Callback(hObject, eventdata, handles)
% hObject    handle to OKTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global port

auxport = get(handles.PortTag,'String');
port = strcat('COM',auxport);


% --- Executes on button press in stateTag.
function stateTag_Callback(hObject, eventdata, handles)
% hObject    handle to stateTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'s','uchar');
stateRS

LSTA = LAUX;
set(handles.stateText,'String',LSTA);

cerrarRS


% --- Executes on button press in userMTag.
function userMTag_Callback(hObject, eventdata, handles)
% hObject    handle to userMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'u','uchar');

cerrarRS

clear all; close all;clc; usertMode;
%usertMode;


% --- Executes on button press in GPSModeTag.
function GPSModeTag_Callback(hObject, eventdata, handles)
% hObject    handle to GPSModeTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'l','uchar');

cerrarRS

clear all; close all;clc; TIEMPOREAL;
%TIEMPOREAL;


% --- Executes on button press in uMTag.
function uMTag_Callback(hObject, eventdata, handles)
% hObject    handle to uMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'u','uchar');

cerrarRS

guidata(hObject,handles);

% --- Executes on button press in wMTag.
function wMTag_Callback(hObject, eventdata, handles)
% hObject    handle to wMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'w','uchar');

cerrarRS

guidata(hObject,handles);
```

```
% --- Executes on button press in lMTag.
function lMTag_Callback(hObject, eventdata, handles)
% hObject    handle to lMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'l','uchar');

cerrarRS

guidata(hObject,handles);
```

10.5. Código programa userMode.m

```
function varargout = usertMode(varargin)
% USERTMODE MATLAB code for usertMode.fig
%      USERTMODE, by itself, creates a new USERTMODE or raises the existing
%      singleton*.
%
%      H = USERTMODE returns the handle to a new USERTMODE or the handle to
%      the existing singleton*.
%
%      USERTMODE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in USERTMODE.M with the given input arguments.
%
%      USERTMODE('Property','Value',...) creates a new USERTMODE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before usertMode_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to usertMode_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help usertMode

% Last Modified by GUIDE v2.5 24-Jun-2012 17:04:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @usertMode_OpeningFcn, ...
                   'gui_OutputFcn',  @usertMode_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before usertMode is made visible.
function usertMode_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to usertMode (see VARARGIN)

% Choose default command line output for usertMode
handles.output = hObject;
```

```
%%%% Iicialización globales %%%%

global port
port = 'COM4'; %puerto por defecto

global stop
global send

stop = 0;
send = 0;

handles.count=1;
set(handles.ConTag,'String','No Connect');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Inicialización imágenes corporativas %%%%

axes(handles.axes1)
background = imread('euitt','jpg');
axis off;
imshow(background);

axes(handles.axes2)
background = imread('indra','jpg');
axis off;
imshow(background);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes usertMode wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = usertMode_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in OpenSTag.
function OpenSTag_Callback(hObject, eventdata, handles)
% hObject    handle to OpenSTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


i=handles.count;
j=1;

global stop
global send

handles.chattext=[];
ext={};
abrirRS
recep = [];
up=[];

while ~stop

    reloj=clock;
    if i>16
        i=16;

        up = get(handles.Chat2,'String');
        set(handles.Chat1,'String',up);
        up = get(handles.Chat3,'String');
        set(handles.Chat2,'String',up);
        up = get(handles.Chat4,'String');
```

```
    set(handles.Chat3,'String',up);
    up = get(handles.Chat5,'String');
    set(handles.Chat4,'String',up);
    up = get(handles.Chat6,'String');
    set(handles.Chat5,'String',up);
    up = get(handles.Chat7,'String');
    set(handles.Chat6,'String',up);
    up = get(handles.Chat8,'String');
    set(handles.Chat7,'String',up);
    up = get(handles.Chat9,'String');
    set(handles.Chat8,'String',up);
    up = get(handles.Chat10,'String');
    set(handles.Chat9,'String',up);
    up = get(handles.Chat11,'String');
    set(handles.Chat10,'String',up);
    up = get(handles.Chat12,'String');
    set(handles.Chat11,'String',up);
    up = get(handles.Chat13,'String');
    set(handles.Chat12,'String',up);
    up = get(handles.Chat14,'String');
    set(handles.Chat13,'String',up);
    up = get(handles.Chat15,'String');
    set(handles.Chat14,'String',up);
    up = get(handles.Chat16,'String');
    set(handles.Chat15,'String',up);
    set(handles.Chat16,'String','');
end

if send
    text = get(handles.enterChat,'String');

    fwrite(PS,'<','uchar');
    while j<=length(text)
        fwrite(PS,text(j),'uchar');
        j=j+1;
    end
    fwrite(PS,'>','uchar');
    j=1;

    text = strcat('S: ',text,'   Time',num2str(reloj(4)),':',num2str(reloj(5)));
    ext(i) = {text};
    set(handles.enterChat,'String','');

    switch i
        case 1
            set(handles.Chat1,'String',text);
        case 2
            set(handles.Chat2,'String',text);
        case 3
            set(handles.Chat3,'String',text);
        case 4
            set(handles.Chat4,'String',text);
        case 5
            set(handles.Chat5,'String',text);
        case 6
            set(handles.Chat6,'String',text);
        case 7
            set(handles.Chat7,'String',text);
        case 8
            set(handles.Chat8,'String',text);
        case 9
            set(handles.Chat9,'String',text);
        case 10
            set(handles.Chat10,'String',text);
        case 11
            set(handles.Chat11,'String',text);
        case 12
            set(handles.Chat12,'String',text);
        case 13
            set(handles.Chat13,'String',text);
        case 14
            set(handles.Chat14,'String',text);
        case 15
            set(handles.Chat15,'String',text);
        case 16
            set(handles.Chat16,'String',text);
    end
```

```
        i=i+1;
        send = 0;
    else
        userRS
        recep = LAUX;

        if ~isempty(recep)
            recep = strcat('R: ',recep,' Time',num2str(reloj(4)),':',num2str(reloj(5)));
            ext(i) = {recep};
            switch i
                case 1
                    set(handles.Chat1,'String',recep);
                case 2
                    set(handles.Chat2,'String',recep);
                case 3
                    set(handles.Chat3,'String',recep);
                case 4
                    set(handles.Chat4,'String',recep);
                case 5
                    set(handles.Chat5,'String',recep);
                case 6
                    set(handles.Chat6,'String',recep);
                case 7
                    set(handles.Chat7,'String',recep);
                case 8
                    set(handles.Chat8,'String',recep);
                case 9
                    set(handles.Chat9,'String',recep);
                case 10
                    set(handles.Chat10,'String',recep);
                case 11
                    set(handles.Chat11,'String',recep);
                case 12
                    set(handles.Chat12,'String',recep);
                case 13
                    set(handles.Chat13,'String',recep);
                case 14
                    set(handles.Chat14,'String',recep);
                case 15
                    set(handles.Chat15,'String',recep);
                case 16
                    set(handles.Chat16,'String',recep);
            end
            i=i+1;
        end
    end

    set(handles.ConTag,'String','Connect');
    close(figure(1));
    clc;

end

stop=0;
set(handles.ConTag,'String','No Connect');
handles.chattext = ext;
handles.count=i;
ext
cerrarRS

guidata(hObject, handles);


% --- Executes on button press in CloseSTag.
function CloseSTag_Callback(hObject, eventdata, handles)
% hObject    handle to CloseSTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global stop

stop = ~stop;

guidata(hObject,handles);
```

```
function PortTag_Callback(hObject, eventdata, handles)
% hObject    handle to PortTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of PortTag as text
%        str2double(get(hObject,'String')) returns contents of PortTag as a double


% --- Executes during object creation, after setting all properties.
function PortTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PortTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in OKTag.
function OKTag_Callback(hObject, eventdata, handles)
% hObject    handle to OKTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global port

auxport = get(handles.PortTag,'String');
port = strcat('COM',auxport);

guidata(hObject, handles);


% --- Executes on button press in stateTag.
function stateTag_Callback(hObject, eventdata, handles)
% hObject    handle to stateTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'s','uchar');
stateRS

LSTA = LAUX;
set(handles.stateText,'String',LSTA);

cerrarRS

guidata(hObject, handles);



function enterChat_Callback(hObject, eventdata, handles)
% hObject    handle to enterChat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of enterChat as text
%        str2double(get(hObject,'String')) returns contents of enterChat as a double
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function enterChat_CreateFcn(hObject, eventdata, handles)
% hObject    handle to enterChat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in sendTag.
function sendTag_Callback(hObject, eventdata, handles)
% hObject    handle to sendTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global send

send = ~send;

guidata(hObject, handles);


% --- Executes on button press in waitMTag.
function waitMTag_Callback(hObject, eventdata, handles)
% hObject    handle to waitMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'w','uchar');

cerrarRS

clear all; close all;clc; waitMode;

% --- Executes on button press in GPSMTag.
function GPSMTag_Callback(hObject, eventdata, handles)
% hObject    handle to GPSMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'l','uchar');

cerrarRS

clear all; close all;clc; TIEMPOREAL;

% --- Executes on button press in CloseCOM.
function CloseCOM_Callback(hObject, eventdata, handles)
% hObject    handle to CloseCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global i
i=17;
delete(instrfindall)


% --- Executes on button press in uMTag.
function uMTag_Callback(hObject, eventdata, handles)
% hObject    handle to uMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'u','uchar');

cerrarRS

guidata(hObject,handles);

% --- Executes on button press in wMTag.
function wMTag_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to wMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'w','uchar');

cerrarRS

guidata(hObject,handles);

% --- Executes on button press in lMTag.
function lMTag_Callback(hObject, eventdata, handles)
% hObject    handle to lMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'l','uchar');

cerrarRS

guidata(hObject,handles);


% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in clrChatTag.
function clrChatTag_Callback(hObject, eventdata, handles)
% hObject    handle to clrChatTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

i=1;
handles.count=1;

while i<=16
    switch i
        case 1
            set(handles.Chat1,'String','');
        case 2
            set(handles.Chat2,'String','');
        case 3
            set(handles.Chat3,'String','');
        case 4
            set(handles.Chat4,'String','');
        case 5
            set(handles.Chat5,'String','');
        case 6
            set(handles.Chat6,'String','');
        case 7
            set(handles.Chat7,'String','');
        case 8
            set(handles.Chat8,'String','');
        case 9
            set(handles.Chat9,'String','');
        case 10
            set(handles.Chat10,'String','');
        case 11
            set(handles.Chat11,'String','');
        case 12
            set(handles.Chat12,'String','');
        case 13
            set(handles.Chat13,'String','');
        case 14
            set(handles.Chat14,'String','');
        case 15
            set(handles.Chat15,'String','');
```

```
        case 16
            set(handles.Chat16,'String','');
    end
    i=i+1;
end

guidata(hObject,handles);



function textfile_Callback(hObject, eventdata, handles)
% hObject    handle to textfile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textfile as text
%        str2double(get(hObject,'String')) returns contents of textfile as a double


% --- Executes during object creation, after setting all properties.
function textfile_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textfile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in savetxtTag.
function savetxtTag_Callback(hObject, eventdata, handles)
% hObject    handle to savetxtTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.chattext
nametxt = get(handles.textfile,'String');
nametxt = strcat(nametxt,'.txt');

fileID = fopen(nametxt,'w');
for i=1:length(handles.chattext)

    fprintf(fileID,'%s\r\n',handles.chattext{i});

end;

fclose(fileID);
```

10.6. Código programa TIEMPOREAL.m (GPS Mode)

```
function varargout = TIEMPOREAL(varargin)
% TIEMPOREAL MATLAB code for TIEMPOREAL.fig
%      TIEMPOREAL, by itself, creates a new TIEMPOREAL or raises the existing
%      singleton*.
%
%      H = TIEMPOREAL returns the handle to a new TIEMPOREAL or the handle to
%      the existing singleton*.
%
%      TIEMPOREAL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TIEMPOREAL.M with the given input arguments.
%
%      TIEMPOREAL('Property','Value',...) creates a new TIEMPOREAL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before TIEMPOREAL_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to TIEMPOREAL_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
```

```
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help TIEMPOREAL

% Last Modified by GUIDE v2.5 08-Aug-2012 14:30:42

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TIEMPOREAL_OpeningFcn, ...
                   'gui_OutputFcn',  @TIEMPOREAL_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before TIEMPOREAL is made visible.
function TIEMPOREAL_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to TIEMPOREAL (see VARARGIN)

% Choose default command line output for TIEMPOREAL
handles.output = hObject;

%%%%%%%%% Inicialización de variables %%%%%%%%

global port %hay que probar port como handles
global manualmode %se definen como global debido a que en tiempo real, es la unica
manera de cambiar
global galttotal
global gspetotal
global gtemtotal
global galtsec
global gspesec
global gtemsec
port = 'COM4'; %puerto por defecto

handles.graph = 0;
handles.seconds = 60;  %tiempo por defecto
handles.secaux = 60;  %tiempo por defecto
handles.manual = 0;
set(handles.UTCinitTag,'visible','off');
set(handles.UTCendTag,'visible','off');
set(handles.DateTag,'visible','off');
manualmode = 0;
galttotal = 0;
gspetotal = 0;
gtemtotal = 0;
galtsec = 0;
gspesec = 0;
gtemsec = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Inicialización imágenes corporativas %%%%

axes(handles.axes2)
background = imread('euitt','jpg');
axis off;
imshow(background);

axes(handles.axes3)
background = imread('indra','jpg');
```

```
axis off;
imshow(background);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
axes(handles.axes1)

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes TIEMPOREAL wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = TIEMPOREAL_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in ShowLiveTag.
function ShowLiveTag_Callback(hObject, eventdata, handles)
% hObject    handle to ShowLiveTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global ALT
global SPE
global SPEK
global TEMP
global UTC
global LAT
global HNS
global LON
global HEW
global DATE
global TEXT
global manualmode
global stopmode
global galttotal
global gspetotal
global gtemtotal
global galtsec
global gspesec
global gtemsec
global linkerror
global ee

ALT = [];  %almacena altitud
SPE = [];  %almacena velocidad
SPEK = []; %almacena velocidad nudos
TEMP = []; %almacena temperatura
UTC = [];  %almacena hora
LAT = [];  %almacena latitud
HNS = [];  %almacena N/S
LON = [];  %almacena longitud
HEW = [];  %almacena E/W
DATE = []; %almacena fecha
TEXT = []; %almacena texto
stopmode = 0;

seconds = 1;

abrirRS

while seconds <= handles.seconds
    leerRS

    ALT = [ALT LALT];
    SPE = [SPE LSPE];
    SPEK = [SPEK LSPEK];
    TEMP = [TEMP LTEMP];
    UTC = [UTC LUTC];
```

```
LAT = [LAT LLAT];
HNS = [HNS; LHNS];
LON = [LON LLON];
HEW = [HEW; LHEW];
DATE = [DATE LDATE];
% Se recogen los valores en variables globales, para poder
% representarlas en tiempo real

seconds = length(ALT); % Para actualizar en caso de pérdida de conexión
figure(1);

SpeStr = num2str(LSPE);
set(handles.velnumTag,'String',SpeStr);
SpekStr = num2str(LSPEK);
set(handles.velknumTag,'String',SpekStr);
AltStr = num2str(LALT);
set(handles.altnumTag,'String',AltStr);
TempStr = num2str(LTEMP);
set(handles.tempnumTag,'String',TempStr);
LatStr = num2str(LLAT);
set(handles.latnumTag,'String',LatStr);
LonStr = num2str(LLON);
set(handles.lonnumTag,'String',LonStr);
set(handles.latletTag,'String',LHNS);
set(handles.lonletTag,'String',LHEW);

if galttotal
    subplot(2,3,1)
    plot(ALT)
    title('Altitude')
    xlabel('Seconds')
    ylabel('Meters')
    grid on
    if seconds >= 2
        axis([1 seconds min(ALT)-15 max(ALT)+15])
    end
else
    subplot(2,3,1)
    plot(0)
    set(subplot(2,3,1),'visible','off');
end
if gspetotal
    subplot(2,3,2)
    plot(SPE)
    title('Speed')
    xlabel('Seconds')
    ylabel('Km/h')
    grid on
    if seconds >= 2
        axis([1 seconds (max((min(SPE)-5),-0.5)) (max(SPE)+5)])
    end
else
    subplot(2,3,2)
    plot(0)
    set(subplot(2,3,2),'visible','off');
end
if gtemtotal
    subplot(2,3,3)
    plot(TEMP)
    title('Temperature')
    xlabel('Seconds')
    ylabel('°C')
    grid on
    if seconds >= 2
        axis([1 seconds (min(TEMP)-1) (max(TEMP)+1)])
    end
else
    subplot(2,3,3)
    plot(0)
    title('Disable')
    set(subplot(2,3,3),'visible','off');
end
if galtsec
    subplot(2,3,4)
    if seconds>1
        if ALT(seconds)>ALT(seconds-1)
            plot(ALT,'r :')
```

```
            elseif ALT(seconds)<ALT(seconds-1)
                plot(ALT,'g :')
            else
                plot(ALT)
            end
        end
        title('Altitude (Last seconds)')
        xlabel('Seconds')
        ylabel('Meters')
        grid on
        if seconds >= 2 && seconds < 11
            axis([1 seconds min(ALT)-15 max(ALT)+15])
        elseif seconds >= 11
            axis([(seconds-10)          seconds          (min(ALT(seconds-10:seconds))-15)
(max(ALT(seconds-10:seconds))+15)])
        end
    else
        subplot(2,3,4)
        plot(0)
        set(subplot(2,3,4),'visible','off');
    end
    if gspesec
        subplot(2,3,5)
        if seconds>1
            if SPE(seconds)>SPE(seconds-1)
                plot(SPE,'r :') %rojo sube valor
            else
                if SPE(seconds)<SPE(seconds-1)
                    plot(SPE,'g :') %verde baja valor
                else
                    plot(SPE) %azul, valor igual
                end
            end
        end
        title('Speed (Last seconds)')
        xlabel('Seconds')
        ylabel('Km/h')
        grid on
        if seconds >= 2 && seconds < 11
            axis([1 seconds (max((min(SPE)-5),-0.5)) (max(SPE)+5)])
        elseif seconds >= 11
            axis([(seconds-10)      seconds      max(min(SPE(seconds-10:seconds))-5,-0.5)
(max(SPE(seconds-10:seconds))+5)])
        end
    else
        subplot(2,3,5)
        plot(0)
        set(subplot(2,3,5),'visible','off');
    end
    if gtemsec
        subplot(2,3,6)
        if seconds>1
            if TEMP(seconds)>TEMP(seconds-1)
                plot(TEMP,'r :') %rojo sube valor
            else
                if TEMP(seconds)<TEMP(seconds-1)
                    plot(TEMP,'g :') %verde baja valor
                else
                    plot(TEMP) %azul, valor igual
                end
            end
        end
        title('Temperature (Last seconds)')
        xlabel('Seconds')
        ylabel('°C')
        grid on
        if seconds >= 2 && seconds < 11
            axis([1 seconds (min(TEMP)-1) (max(TEMP)+1)])
        elseif seconds >= 11
            axis([(seconds-10)          seconds          (min(TEMP(seconds-10:seconds))-1)
(max(TEMP(seconds-10:seconds))+1)])
        end
    else
        subplot(2,3,6)
        plot(0)
        set(subplot(2,3,6),'visible','off');
    end
```

```
    clc;

    seconds = seconds+1;


    if stopmode
        seconds = handles.seconds+1;
    elseif manualmode
        handles.seconds = seconds+1;
    end

    if ee == 1
        handles.seconds = seconds;
        fprintf('Error fatal, la toma de datos se corto');
    end
end

% Se utiliza para deshacer en caso de parada
if handles.manual
    manualmode = handles.manual;
elseif stopmode
    stopmode = 0;
end

handles.seconds = handles.secaux;

% Se utiliza al cargar el workspace
handles.ALT = ALT;
handles.SPE = SPE;
handles.SPEK = SPEK;
handles.TEMP = TEMP;
handles.UTC = UTC;
handles.LAT = LAT;
handles.HNS = HNS;
handles.LON = LON;
handles.HEW = HEW;
handles.DATE = DATE;
handles.TEXT = TEXT;

cerrarRS
guidata(hObject, handles);


% --- Executes on button press in AnalizeTag.
function AnalizeTag_Callback(hObject, eventdata, handles)
% hObject    handle to AnalizeTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global ALT
global SPE
global TEMP
global UTC
global DATE

handles.optgraph=get(handles.selectmenu,'Value');
switch handles.optgraph
    case 1
        plot(ALT);
        title('Altitude')
        xlabel('Seconds')
        ylabel('Meters')
        axis([1 length(ALT) min(ALT)-15 max(ALT)+15])
    case 2
        plot(SPE);
        title('Speed')
        xlabel('Seconds')
        ylabel('Km/h')
        axis([1 length(SPE) max((min(SPE)-5),-0.5) max(SPE)+5])
    case 3
        plot(TEMP);
        title('Temperature')
        xlabel('Seconds')
        ylabel('°C')
        axis([1 length(TEMP)  min(TEMP)-1 max(TEMP)+1])
end
```

```
grid on

set(handles.UTCinitTag,'visible','on');
set(handles.UTCendTag,'visible','on');
set(handles.DateTag,'visible','on');

UTCStr = num2str(UTC(1));
UTCStr = strcat(UTCStr(1:2),':',UTCStr(3:4),':',UTCStr(5:6));
set(handles.UTCinitTag,'String',UTCStr);
UTCStr = num2str(UTC(length(UTC)));
UTCStr = strcat(UTCStr(1:2),':',UTCStr(3:4),':',UTCStr(5:6));
set(handles.UTCendTag,'String',UTCStr);
DATEStr = num2str(DATE(1));
if length(DATEStr) == 5
    DATEStr = strcat(DATEStr(1),'/',DATEStr(2:3),'/',DATEStr(4:5));
else
    DATEStr = strcat(DATEStr(1:2),'/',DATEStr(3:4),'/',DATEStr(5:6));
end
set(handles.DateTag,'String',DATEStr);


guidata(hObject, handles);


% --- Executes on button press in AltTotTag.
function AltTotTag_Callback(hObject, eventdata, handles)
% hObject    handle to AltTotTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of AltTotTag

global galttotal
% Se declara como global, para poder modificar su
% valor durante la ejecución del bucle

galttotal = get(hObject,'Value');

guidata(hObject,handles);


% --- Executes on button press in AltSecTag.
function AltSecTag_Callback(hObject, eventdata, handles)
% hObject    handle to AltSecTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of AltSecTag

global galtsec

galtsec = get(hObject,'Value');

guidata(hObject,handles);


% --- Executes on button press in SpeTotTag.
function SpeTotTag_Callback(hObject, eventdata, handles)
% hObject    handle to SpeTotTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of SpeTotTag

global gspetotal

gspetotal = get(hObject,'Value');

guidata(hObject,handles);


% --- Executes on button press in SpeSecTag.
function SpeSecTag_Callback(hObject, eventdata, handles)
% hObject    handle to SpeSecTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of SpeSecTag

global gspesec

gspesec = get(hObject,'Value');

guidata(hObject,handles);


% --- Executes on button press in TemTotTag.
function TemTotTag_Callback(hObject, eventdata, handles)
% hObject    handle to TemTotTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of TemTotTag

global gtemtotal

gtemtotal = get(hObject,'Value');

guidata(hObject,handles);


% --- Executes on button press in TemSecTag.
function TemSecTag_Callback(hObject, eventdata, handles)
% hObject    handle to TemSecTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of TemSecTag

global gtemsec

gtemsec = get(hObject,'Value');

guidata(hObject,handles);


% --- Executes on selection change in selectmenu.
function selectmenu_Callback(hObject, eventdata, handles)
% hObject    handle to selectmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns selectmenu contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from selectmenu


% --- Executes during object creation, after setting all properties.
function selectmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to selectmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if             ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function textws_Callback(hObject, eventdata, handles)
% hObject    handle to textws (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textws as text
%        str2double(get(hObject,'String')) returns contents of textws as a double


% --- Executes during object creation, after setting all properties.
function textws_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to textws (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in savews.
function savews_Callback(hObject, eventdata, handles)
% hObject    handle to savews (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

save (get(handles.textws,'String'))
%guardar


% --- Executes on button press in loadws.
function loadws_Callback(hObject, eventdata, handles)
% hObject    handle to loadws (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load (get(handles.textws,'String'))

global ALT
global SPE
global SPEK
global TEMP
global UTC
global LAT
global HNS
global LON
global HEW
global DATE

ALT = handles.ALT;
SPE = handles.SPE;
SPEK = handles.SPEK;
TEMP = handles.TEMP;
UTC = handles.UTC;
LAT = handles.LAT;
HNS = handles.HNS;
LON = handles.LON;
HEW = handles.HEW;
DATE = handles.DATE;


function secondsTag_Callback(hObject, eventdata, handles)
% hObject    handle to secondsTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of secondsTag as text
%        str2double(get(hObject,'String')) returns contents of secondsTag as a double


% --- Executes during object creation, after setting all properties.
function secondsTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to secondsTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in OKsecTag.
```

```
function OKsecTag_Callback(hObject, eventdata, handles)
% hObject    handle to OKsecTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global port
global manualmode

if ~manualmode
sec = str2double(get(handles.secondsTag,'String'));
handles.seconds = sec;
handles.secaux = sec;
end

auxport = get(handles.PortTag,'String');
port = strcat('COM',auxport);

guidata(hObject, handles);



function PortTag_Callback(hObject, eventdata, handles)
% hObject    handle to PortTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of PortTag as text
%        str2double(get(hObject,'String')) returns contents of PortTag as a double


% --- Executes during object creation, after setting all properties.
function PortTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PortTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function UTCinitTag_Callback(hObject, eventdata, handles)
% hObject    handle to UTCinitTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of UTCinitTag as text
%        str2double(get(hObject,'String')) returns contents of UTCinitTag as a double


% --- Executes during object creation, after setting all properties.
function UTCinitTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to UTCinitTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function UTCendTag_Callback(hObject, eventdata, handles)
% hObject    handle to UTCendTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of UTCendTag as text
%        str2double(get(hObject,'String')) returns contents of UTCendTag as a double
```

```
% --- Executes during object creation, after setting all properties.
function UTCendTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to UTCendTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in TrackTag.
function TrackTag_Callback(hObject, eventdata, handles)
% hObject    handle to TrackTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global ALT
global SPE
global TEMP

global LAT
global LON
handles.LAT = LAT;
handles.LON = LON;
global HNS;
global HEW;
handles.HNS = HNS;
handles.HEW = HEW;

auxlat = [];
auxlon = [];

handles.optgraph=get(handles.selectmenu,'Value');
switch handles.optgraph
    case 1
        handles.graph = ALT;
    case 2
        handles.graph = SPE;
    case 3
        handles.graph = TEMP;
end
j=1;
i=1;

% filtra los resultados erróneos
while(i<=length(handles.LAT))
    if handles.HNS(i) == 'N'
        auxlat(j)=handles.LAT(i);
    elseif handles.HNS(i) == 'S'
        auxlat(j)=-handles.LAT(i);
    else
        if j==1
            auxlat(j)='N';
        else
            auxlat(j)=auxlat(j-1);
        end
    end

    if handles.HEW(i) == 'E'
        auxlon(j)=handles.LON(i);
    elseif handles.HEW(i) == 'W'
        auxlon(j)=-handles.LON(i);
    else
        if j==1
            auxlon(j)='W';
        else
            auxlon(j)=auxlon(j-1);
        end
    end
    auxgraph(j)=handles.graph(i);
    j=j+1;
    i=i+1;
```

```
end

switch handles.optgraph
    case 1
        figure(2);
        plot3(auxlon,auxlat,auxgraph);
        title('Track')
        xlabel('Longitude (E=+ W=-)')
        ylabel('Latitude (N=+ S=-)')
        zlabel('Altitude (m)')
        grid on
    case 2
        figure(2);
        plot3(auxlon,auxlat,auxgraph);
        title('Track')
        xlabel('Longitude')
        ylabel('Latitude')
        zlabel('Speed (Km/h)')
        grid on
    case 3
        figure(2);
        plot3(auxlon,auxlat,auxgraph);
        title('Track')
        xlabel('Longitude')
        ylabel('Latitude')
        zlabel('Temperature (°C)')
        grid on
end

guidata(hObject, handles);


% --- Executes on button press in StopTag.
function StopTag_Callback(hObject, eventdata, handles)
% hObject    handle to StopTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global manualmode
global stopmode

if manualmode
    manualmode = ~manualmode;
else
    stopmode = 1;
end

guidata(hObject,handles);

% --- Executes on button press in ManualTag.
function ManualTag_Callback(hObject, eventdata, handles)
% hObject    handle to ManualTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of ManualTag

global manualmode
manualmode = get(hObject,'Value');
handles.manual = get(hObject,'Value');
% handles.manual actua como auxiliar de manualmode, durante el tiempo de
% ejecución de ShowLive

if handles.manual
    set(handles.secondsTag,'enable','off');
    set(handles.SecondsText,'enable','off');
else
    set(handles.secondsTag,'enable','on');
    set(handles.SecondsText,'enable','on');
    handles.seconds = handles.secaux;
end;

guidata(hObject,handles);
```

```
function DateTag_Callback(hObject, eventdata, handles)
% hObject    handle to DateTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of DateTag as text
%        str2double(get(hObject,'String')) returns contents of DateTag as a double


% --- Executes during object creation, after setting all properties.
function DateTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to DateTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function texttf_Callback(hObject, eventdata, handles)
% hObject    handle to texttf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of texttf as text
%        str2double(get(hObject,'String')) returns contents of texttf as a double


% --- Executes during object creation, after setting all properties.
function texttf_CreateFcn(hObject, eventdata, handles)
% hObject    handle to texttf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in txtTag.
function txtTag_Callback(hObject, eventdata, handles)
% hObject    handle to txtTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global ALT
global SPE
global SPEK
global TEMP
global UTC
global LAT
global HNS
global LON
global HEW
global DATE

handles.ALT = ALT;
handles.SPE = SPE;
handles.SPEK = SPEK;
handles.TEMP = TEMP;
handles.UTC = UTC;
handles.LAT = LAT;
handles.HNS = HNS;
handles.LON = LON;
handles.HEW = HEW;
handles.DATE = DATE;

nametxt = get(handles.texttf,'String');
nametxt = strcat(nametxt,'.txt');
```

```
fileID = fopen(nametxt,'w');
for i=1:length(handles.LAT)


fprintf(fileID,'/GPSTEMP,%s,%s,%s,%s,%s',num2str(handles.UTC(i)'),num2str(handles.LAT(i)
'),handles.HNS(i),num2str(handles.LON(i)'),handles.HEW(i));

fprintf(fileID,',%s,M,%s,KM/H,%s,K',num2str(handles.ALT(i)),num2str(handles.SPE(i)'),num
2str(handles.SPEK(i)'));

fprintf(fileID,',%s,C,%s,ID%i\r\n',num2str(handles.TEMP(i)'),num2str(handles.DATE(i)'),i
);

end;

fclose(fileID);


% --- Executes on button press in GPSTag.
function GPSTag_Callback(hObject, eventdata, handles)
% hObject    handle to GPSTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global ALT
global SPE
global SPEK
global TEMP
global UTC
global LAT
global HNS
global LON
global HEW
global DATE

handles.ALT = ALT;
handles.SPE = SPE;
handles.SPEK = SPEK;
handles.TEMP = TEMP;
handles.UTC = UTC;
handles.LAT = LAT;
handles.HNS = HNS;
handles.LON = LON;
handles.HEW = HEW;
handles.DATE = DATE;

nametxt = get(handles.texttf,'String');
nametxt = strcat(nametxt,'.txt');

fileID = fopen(nametxt,'w');
for i=1:length(handles.LAT)


fprintf(fileID,'$GPGGA,%s,%s,%s,%s,%s,,,,%s,M,,,,\r\n',num2str(handles.UTC(i)'),num2str(
handles.LAT(i)'),handles.HNS(i),num2str(handles.LON(i)'),handles.HEW(i),num2str(handles.
ALT(i)));

fprintf(fileID,'$GPRMC,%s,,%s,%s,%s,%s,%s,,%s,,\r\n',num2str(handles.UTC(i)'),num2str(ha
ndles.LAT(i)'),handles.HNS(i),num2str(handles.LON(i)'),handles.HEW(i),num2str(handles.SP
EK(i)),num2str(handles.DATE(i)'));

end;

fclose(fileID);


% --- Executes on button press in CloseCOM.
function CloseCOM_Callback(hObject, eventdata, handles)
% hObject    handle to CloseCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

delete(instrfindall)


% --- Executes on button press in FigureAnTag.
function FigureAnTag_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to FigureAnTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global ALT
global SPE
global TEMP

handles.ALT = ALT;
handles.SPE = SPE;
handles.TEMP = TEMP;

handles.optgraph=get(handles.selectmenu,'Value');
switch handles.optgraph
    case 1
        figure(3);
        plot(handles.ALT);
        title('Altitude')
        xlabel('Seconds')
        ylabel('Meters')
        axis([1 length(ALT) min(ALT)-15 max(ALT)+15])
    case 2
        figure(3);
        plot(handles.SPE);
        title('Speed')
        xlabel('Seconds')
        ylabel('Km/h')
        axis([1 length(SPE) max((min(SPE)-5),-0.5) max(SPE)+5])
    case 3
        figure(3);
        plot(handles.TEMP);
        title('Temperature')
        xlabel('Seconds')
        ylabel('°C')
        axis([1 length(TEMP)  min(TEMP)-1 max(TEMP)+1])
end

grid on

guidata(hObject, handles);


% --- Executes on button press in userMTag.
function userMTag_Callback(hObject, eventdata, handles)
% hObject    handle to userMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'u','uchar');

cerrarRS

clear all; close all;clc; usertMode;
%usertMode;

% --- Executes on button press in waitMTag.
function waitMTag_Callback(hObject, eventdata, handles)
% hObject    handle to waitMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'w','uchar');

cerrarRS

clear all; close all;clc; waitMode;


% --- Executes on button press in stateTag.
function stateTag_Callback(hObject, eventdata, handles)
% hObject    handle to stateTag (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'s','uchar');
stateRS

LSTA = LAUX;
set(handles.stateText,'String',LSTA);

cerrarRS


% --- Executes on button press in uMTag.
function uMTag_Callback(hObject, eventdata, handles)
% hObject    handle to uMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'u','uchar');

cerrarRS

guidata(hObject,handles);

% --- Executes on button press in wMTag.
function wMTag_Callback(hObject, eventdata, handles)
% hObject    handle to wMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'w','uchar');

cerrarRS

guidata(hObject,handles);

% --- Executes on button press in lMTag.
function lMTag_Callback(hObject, eventdata, handles)
% hObject    handle to lMTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

abrirRS

fwrite(PS,'&','uchar');
fwrite(PS,'l','uchar');

cerrarRS

guidata(hObject,handles);



function velnumTag_Callback(hObject, eventdata, handles)
% hObject    handle to velnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of velnumTag as text
%        str2double(get(hObject,'String')) returns contents of velnumTag as a double


% --- Executes during object creation, after setting all properties.
function velnumTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to velnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function velknumTag_Callback(hObject, eventdata, handles)
% hObject    handle to velknumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of velknumTag as text
%        str2double(get(hObject,'String')) returns contents of velknumTag as a double


% --- Executes during object creation, after setting all properties.
function velknumTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to velknumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function altnumTag_Callback(hObject, eventdata, handles)
% hObject    handle to altnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of altnumTag as text
%        str2double(get(hObject,'String')) returns contents of altnumTag as a double


% --- Executes during object creation, after setting all properties.
function altnumTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to altnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function tempnumTag_Callback(hObject, eventdata, handles)
% hObject    handle to tempnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tempnumTag as text
%        str2double(get(hObject,'String')) returns contents of tempnumTag as a double


% --- Executes during object creation, after setting all properties.
function tempnumTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tempnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function latnumTag_Callback(hObject, eventdata, handles)
% hObject    handle to latnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of latnumTag as text
%        str2double(get(hObject,'String')) returns contents of latnumTag as a double


% --- Executes during object creation, after setting all properties.
function latnumTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to latnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function lonnumTag_Callback(hObject, eventdata, handles)
% hObject    handle to lonnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lonnumTag as text
%        str2double(get(hObject,'String')) returns contents of lonnumTag as a double


% --- Executes during object creation, after setting all properties.
function lonnumTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lonnumTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function latletTag_Callback(hObject, eventdata, handles)
% hObject    handle to latletTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of latletTag as text
%        str2double(get(hObject,'String')) returns contents of latletTag as a double


% --- Executes during object creation, after setting all properties.
function latletTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to latletTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function lonletTag_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to lonletTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lonletTag as text
%        str2double(get(hObject,'String')) returns contents of lonletTag as a double


% --- Executes during object creation, after setting all properties.
function lonletTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lonletTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

10.7. Código programa abrirRS.m

```
%function abrirRS

global port

PS = serial(port);
set(PS,'Baudrate',4800); % se configura la velocidad a 4800 Baudios
%set(PS,'Baudrate',9600); % se configura la velocidad a 9600 Baudios
set(PS,'StopBits',1); % se configura bit de parada a uno
set(PS,'DataBits',8); % se configura que el dato es de 8 bits, debe estar entre 5 y 8
set(PS,'Parity','none'); % se configura sin paridad
set(PS,'Terminator','CR/LF');% "c" caracter con que finaliza el envío
set(PS,'OutputBufferSize',1); % "n" es el número de bytes a enviar
set(PS,'InputBufferSize' ,1); % "n" es el número de bytes a recibir
set(PS,'Timeout',1); % 5 segundos de tiempo de espera

fopen(PS);
```

10.8. Código programa cerrarRS.m

```
%function cerrarRS

fclose(PS);
delete(PS);
clear PS;
```

10.9. Código programa leerRs.m

```
%function leerRS

LIN = [];  % array que lee valores
LALT = [];  % array que almacena la altitud
LAUX = [];  % array auxiliar para conseguir valor numérico
LAUX2 = [];  % variable auxiliar para conseguir valor numérico
LSPE = [];  % array que almacena la velocidad
LTEMP = [];  % array que almacena la temperatura
LUTC = [];  % array que almacena el horario UTC
LLAT = [];  % array que almacena la latitud
LHNS = []; % array que almacena hemisferio N=1 S=-1
LLON = []; % array que almacena la longitud
LHEW = []; % array que almacena hemisferio E=1 W=-1
LDATE = []; % array que almacena la fecha
LSPEK = []; % array que almacena la velocidad en nudos
```

```
p = 0; %numero de comas leidas
t = 0; %numero de trama

error = 0;
ceros = 0;
e = [];
eee = 0;
s=0; %segundos de pérdida enlace
global linkerror


while t < 3
LIN = fread(PS,1,'uchar');
LIN = char(LIN);

if LIN == '$'
    for j = 1:5
        LIN = [LIN fread(PS,1,'uchar')];
    end
    if LIN == '$GPGGA'
        if t == 0
        while p < 9

            switch p
                case 1
                    LIN = fread(PS,1,'uchar');
                    LIN = char(LIN);
                    LAUX = LIN;
                    while LIN ~= ','
                        LIN = fread(PS,1,'uchar');
                        LAUX2 = LIN;
                        LIN = char(LIN);
                        LAUX = [LAUX LAUX2];
                    end;
                    LAUX = char(LAUX);
                    aux                                                                    =
[str2double(LAUX(1:2));str2double(LAUX(3:4));str2double(LAUX(5:6))];
                    LUTC = str2double(LAUX); %Lectura hora UTC
                    if LUTC>235959
                        LUTC = 0;
                    elseif LUTC<0
                        LUTC = 0;
                    end
                    LAUX = [];
                    p = p+1;
                case 2
                    LIN = fread(PS,1,'uchar');
                    LIN = char(LIN);
                    LAUX = LIN;
                    while LIN ~= ','
                        LIN = fread(PS,1,'uchar');
                        LAUX2 = LIN;
                        LIN = char(LIN);
                        LAUX = [LAUX LAUX2];
                    end
                    LAUX = char(LAUX);
                    LLAT = str2double(LAUX); %Lectura latitud
                    LAUX = [];
                    p = p+1;
                case 3
                    LIN = char(fread(PS,1,'uchar'));
                    LHNS = LIN; %Lectura hemisferio norte-sur
                    while LIN ~= ','
                        LIN = fread(PS,1,'uchar');
                        LAUX2 = LIN;
                        LIN = char(LIN);
                        LAUX = [LAUX LAUX2];
                    end;
                    p = p+1;
                case 4
                    LIN = fread(PS,1,'uchar');
                    LIN = char(LIN);
                    LAUX = LIN;
                    while LIN ~= ','
                        LIN = fread(PS,1,'uchar');
                        LAUX2 = LIN;
```

```
                    LIN = char(LIN);
                    LAUX = [LAUX LAUX2];
                end;
                LAUX = char(LAUX);
                LLON = str2double(LAUX); %Lectura longitud
                LAUX = [];
                p = p+1;
            case 5
                LIN = fread(PS,1,'uchar');
                LIN = char(LIN);
                LHEW = LIN; %Lectura hemisferio este-oeste
                while LIN ~= ','
                    LIN = fread(PS,1,'uchar');
                    LAUX2 = LIN;
                    LIN = char(LIN);
                    LAUX = [LAUX LAUX2];
                end;
                p = p+1;
        end

        if p<1 || p>=6
            LIN = fread(PS,1,'uchar');
            LIN = char(LIN);
            if LIN == ','
                p = p+1;
            end
        end
    end
    %Lectura de la 9na coma, siguiente lectura altitud

    LIN = fread(PS,1,'uchar');
    LIN = char(LIN);
    LAUX = LIN;
    while LIN ~= ','
        LIN = fread(PS,1,'uchar');
        LAUX2 = LIN;
        LIN = char(LIN);
        LAUX = [LAUX LAUX2];
    end;

    LAUX = char(LAUX);
    LALT = str2double(LAUX); %Lectura altitud
    if LALT > 4000
        LALT = 0;
    elseif LALT < -100
        LALT = 0;
    end
    LAUX = [];
    end;

    %Final primera trama GPS
    t = t+1;

elseif LIN == '$GPRMC'
    if t == 1 %Para evitar malas lecturas
    while p<7
        LIN = fread(PS,1,'uchar');
        LIN = char(LIN);
        if LIN == ','
        p = p+1;
        end;
    end;

    LIN = fread(PS,1,'uchar');
    LIN = char(LIN);
    LAUX = LIN;
    while LIN ~= ','
        LIN = fread(PS,1,'uchar');
        LAUX2 = LIN;
        LIN = char(LIN);
        LAUX = [LAUX LAUX2];
    end;
    LAUX = char(LAUX);
    LSPEK = str2double(LAUX); %nudos, lectura de velocidad
    LSPE = str2double(LAUX)*1.852; %km/h, lectura de velocidad
    if LSPE > 180
        LSPE = 0;
```

```
            LSPEK = 0;
        elseif LSPE < 0
            LSPE = 0;
            LSPEK = 0;
        end
        LAUX = [];

        p = p+1;
        while p<9
            LIN = fread(PS,1,'uchar');
            LIN = char(LIN);
            if LIN == ','
            p = p+1;
            end;
        end;
        LIN = fread(PS,1,'uchar');
        LIN = char(LIN);
        LAUX = LIN;
        while LIN ~= ','
            LIN = fread(PS,1,'uchar');
            LAUX2 = LIN;
            LIN = char(LIN);
            LAUX = [LAUX LAUX2];
        end;
        LAUX = char(LAUX);
        LDATE = str2double(LAUX); %Lectura fecha
        if LDATE > 311299
            LDATE = 0;
        elseif LDATE < 10100
            LDATE = 0;
        end
        LAUX = [];

        %Final cuarta trama GPS, segunda utilizada
        t = t+1;
        end;
    end;


elseif LIN == '/'
    if t == 2
        for j = 1:4
            LIN = [LIN fread(PS,1,'uchar')];
        end

        if LIN == '/TEMP'
            while p < 1
                LIN = fread(PS,1,'uchar');
                LIN = char(LIN);
                if LIN == ','
                    p = p+1;
                end
            end

            t = t+1;
            LIN = fread(PS,1,'uchar');
            LIN = char(LIN);
            LAUX = LIN;
            while LIN ~= ','
                LIN = fread(PS,1,'uchar');
                LAUX2 = LIN;
                LIN = char(LIN);
                LAUX = [LAUX LAUX2];
            end
            LAUX = char(LAUX);
            LTEMP = str2double(LAUX); %lectura temperatura
            if LTEMP > 80
                LTEMP = 0;
            elseif LTEMP < -40
                LTEMP = 0;
            end
            LAUX = [];
        end
    end
    %final trama temperatura
end
```

```
p = 0; %reiniación de variable que lee los puntos
LIN = [];
end

if linkerror == 0
    linkerror = aux;
else
    e = aux-linkerror; %error
    if e(1) <0
        e(1) = e(1)+24;
    end
    s = e(1)*3600+e(2)*60+e(3)-1-1; %error expresado en segundos
    %el ultimo 1 se resta, por la comprobación de isempty

    linkerror = aux;
end

if s>0
    e = s+1;
    ceros = 1;
    CERO = [0];
    CEROC = char('A');
    for i=1:s
        CERO = [CERO, 0];
        CEROC = [CEROC; char('A')];
    end
end

if isempty(LALT)
    LALT = 0;
end
if isempty(LSPE)
    LSPE = 0;
end
if isempty(LTEMP)
    LTEMP = 0;
end
if isempty(LUTC)
    LUTC = 0;
end
if isempty(LLAT)
    LLAT = 0;
end
if isempty(LHNS)
    LHNS = 'A';
end
if isempty(LLON)
    LLON = 0;
end
if isempty(LHEW)
    LHEW = 'A';
end
if isempty(LDATE)
    LDATE = 0;
end
if isempty(LSPEK)
    LSPEK = 0;
end

if ceros == 1
    LALT = [CERO LALT]
    LSPE = [CERO LSPE]
    LTEMP = [CERO LTEMP]
    LUTC = [CERO LUTC]
    LLAT = [CERO LLAT]
    LHNS = [CEROC; LHNS]
    LLON = [CERO LLON]
    LHEW = [CEROC; LHEW]
    LDATE = [CERO LDATE]
    LSPEK = [CERO LSPEK]
else
    LALT
    LSPE
    LTEMP
    LUTC
    LLAT
    LHNS
```

```
    LLON
    LHEW
    LDATE
    LSPEK
end
```

10.10. Código programa stateRS.m

```
%stateRS

LIN = [];  % array que lee valores
LAUX = [];  % array auxiliar para conseguir valor numérico
LCHAT = []; % array que devuelve el valor de la frase completa


LIN = fread(PS,1,'uchar');
LIN = char(LIN);
LAUX = LIN;
while LIN ~= '&'
    LIN = fread(PS,1,'uchar');
    LIN = char(LIN);
    LAUX = LIN;
end
LAUX=[];%para modo State
while LIN ~= char(10)
    LIN = fread(PS,1,'uchar');
    LAUX2 = LIN;
    LIN = char(LIN);
    LAUX = [LAUX LAUX2];
end
LAUX = char(LAUX);

LAUX
```

10.11. Código programa userRs.m

```
%userRS

LIN = [];  % array que lee valores
LAUX = [];  % array auxiliar para conseguir valor numérico
LCHAT = []; % array que devuelve el valor de la frase completa


LIN = fread(PS,1,'uchar');
LIN = char(LIN);
while LIN ~= '<'
    LIN = fread(PS,1,'uchar');
    LAUX2 = LIN;
    LIN = char(LIN);
end
while LIN ~= '>'
    LIN = fread(PS,1,'uchar');
    LAUX2 = LIN;
    LIN = char(LIN);
    if LAUX2~='>'
        LAUX = [LAUX LAUX2];
    end
end

LAUX = char(LAUX);

LAUX
```